Marin Fotache

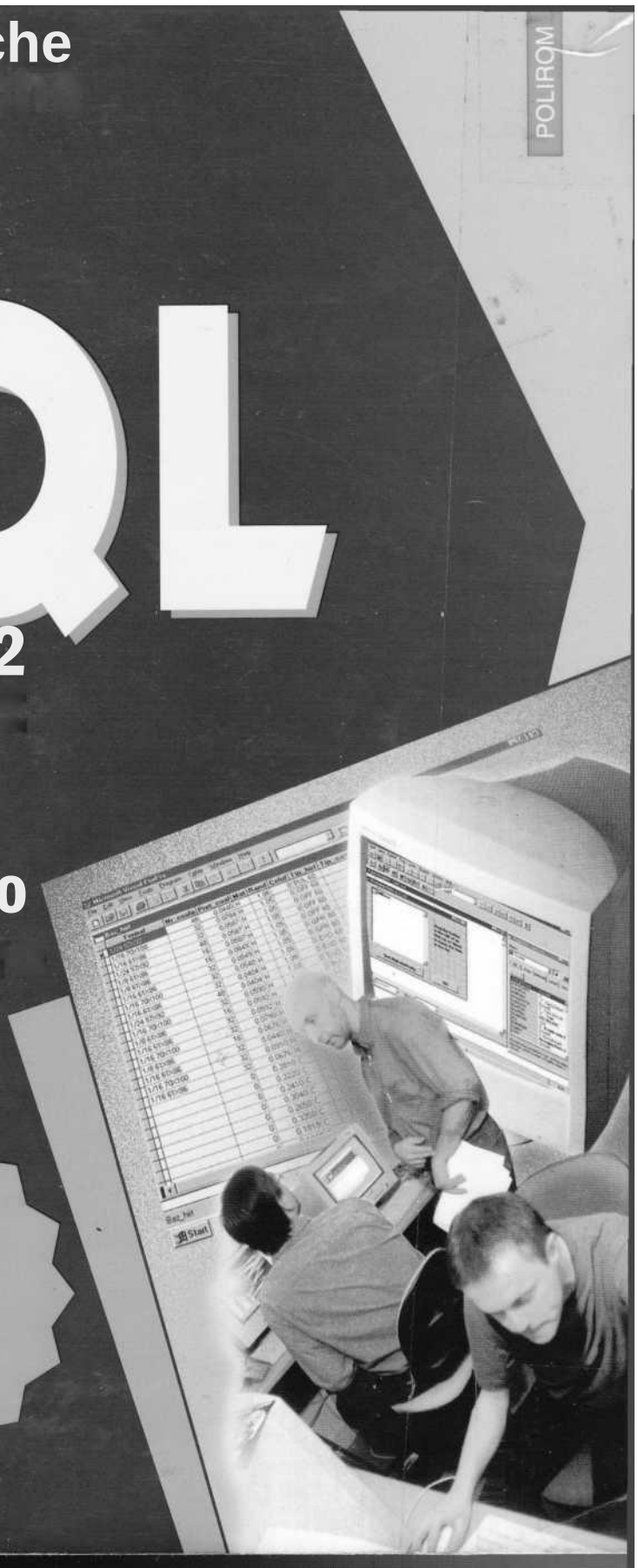
POLIROM

SQL

Dialecte DB2

Oracle
Visual FoxPro

De la
simplu
la
complex



CUVÂNT, OARECUM, ÎNAINTE

Un amic, al cărui succes publicistic îl invidiez de ani buni, m-a convins că, uneori, titlul este la fel de important ca şi conţinutul unei cărţi. Distinsul „cetitor" care intră în librărie sau anticariat, hotărât să nu se atingă de bruma din portmoneu, trebuie capturat cu orice preţ de o copertă viu colorată, de un titlu sprinţar şi convins să cumpere ori, în cel mai rău caz, poate pleca chinuit de păcatul de a fi lăsat cartea în raft (sentimentul se numeşte, în termeni tehnici, frustrare).

Astfel, prima mea grijă a fost găsirea unui titlu straşnic. *SQL pentru to(n)ţi* ar fi fost prea jignitor şi, oricum, nerealist. Nu cred că poţi şti SQL şi să fii, simultan, tont. Vă asigur că dacă vă descurcaţi cu SQL la un nivel acceptabil, sunteţi oricum, dar nu şi tont.

Nici *Totul despre SQL* nu mi s-a părut grozav. în primul rând, deoarece a fost deja folosit de Corina şi Adrian Pascu. Apoi, ar fi fost o exagerare. Un tratat complet despre SQL s-ar întinde pe câteva mii de pagini şi, chiar şi aşa, ar mai rămâne destule puncte neatinse sau tratate superficial, motiv pentru care mi-am reprimat urgent bruma de vocaţie academistă.

Anxios din cale-afară, am profitat de primul vin roşu ieşit în cale şi mi-am provocat prietenii la un *wine-storming.* Le-am povestit problema şi pe dată au început să curgă sugestiile, criticile şi propunerile. *Olecuţă de SQL* e prea moldovenesc. Polirom este o editură cu o bătaie lungă şi largă şi nu s-ar cuveni o regionalizare a ţării tocmai pe criterii SQL-istice.

O primă propunere serioasă a fost *Esenţă de SOL.* Mi-a plăcut, dar parcă prea suna a elită - or, cel puţin prin titlu, cartea trebuia să se adreseze maselor largi de studenţi, informaticieni şi altora dispuşi sau siliţi să lucreze cu bazele de date. încălziţi fiind, ne-am amintit de prestaţia lui Al Pacino din *Parfum de femeie,* aşa că următoarea idee a fost *Parfum de SQL.* Am realizat că exagerasem, deoarece unui cârtitor versat i-ar fi trecut prin cap ideea că mai bine suna *After-shave de SQL.*

Cum apucasem să mă laud, cu câteva luni în urmă, cu rubrica din *Net Report* (*ex-PC Report*), unul dintre meseni a sugerat ca titlul cărţii să fie chiar *Fiola de SQL.* Există deja câteva *mii [zeci!,* l-am corectat eu) de posibili cititori care au auzit de această expresie. M-am opus net (report), din mai multe motive. Nu mi s-a părut corect faţă de cei de la revistă (acum mă suspectez că am avut în vedere şi o eventuală culegere de fiole într-o altă carte, idee care, la acest moment, nu mi pare chiar realistă). Şi apoi, unde s-a mai pomenit ca o fiolă să se întindă pe câteva sute de pagini? Poate *Butoiul cu SQL* - dar şi asta suna ca un atac (bahic) la persoana studentului/informaticianului/ specialistului/ cumpărătorului.

...Şi uite aşa am ajuns la prea puţin sofisticatul titlu *SQL.* Pur şi simplu. Adio regionalisme, esenţe, deodorante şi fiole. Pentru a mai adăuga un pic de semnificaţie şi a ţinti câteva categorii clare de utilizatori/cumpărători ai cărţii, am mai adăugat subtitlul *Dialecte DB2, Oracle şi Visual FoxPro.* Asta e. Să nu ziceţi că titlul n-are imaginaţie. N-aş suporta lovitura...

Cartea de faţă este destinată unor grupuri de utilizatori ce folosesc sau sunt pe cale de a folosi bazele de date. Pentru studenţii specializărilor Informatică Economică şi Birotică (de la Facultatea de Economie şi Administrarea Afacerilor) ar putea fi un ajutor nepreţuit în efortul supraomenesc de a promova anul de studii în care apare disciplina *Baze de date* (glumeam, cred că am, în rândul studenţilor, o rată de sinucideri care l-ar dezamăgi pe Cioran).

Studenţii de alte facultăţi (Informatică, Cibernetică, Calculatoare) o pot folosi chiar dacă-i supără pe profesorii lor titulari.

întrucât am reuşit (până acum) să îmbin lucrul la catedră şi munca patriotică din facultate cu o serie de contracte/participări la realizarea unor aplicaţii (nu puţine au fost eşecurile), îndrăznesc să o recomand şi practicienilor, celor care dezvoltă aplicaţii cu baze de date, dar şi celor care au baze de date implementate de ceva timp şi vor să obţină tot soiul de informaţii şi nu au la dispoziţie un informatician sau sunt certaţi cu acesta.

O lucrare serioasă capătă cu adevărat prestigiu dacă prefaţa se termină cu mulţumiri. întrucât aveam de acasă, ca şi laureaţii premiilor Oscar, o listă cu nume, dar am rătăcit-o pe undeva, nu-mi rămâne decât să le mulţumesc tuturor celor pe care i-am cunoscut, indiferent de media (vizual/ auditiv/e-mail/reviste), relaţii de rudenie, prietenie sau indiferenţă, precum şi celor pe care îi voi cunoaşte până la următoarea carte (când sper să le mulţumesc din nou). Unora vreau să le cer scuze dacă plăcerea *lor* de a mă cunoaşte nu a fost la înălţimea plăcerii *mele* de a-i cunoaşte. Timpul nu e pierdut. Ce-aţi zice să mai încercaţi o dată, cu cartea aceasta în faţă?

Mă simt obligat să mărturisesc că o parte din materialul bibliografic parcurs în anii începuturilor mele în SQL (şi acum învăţ SQL, deşi nu mai am sporul de altădată) a fost preluat de pe Internet, precum şi prin fondurile Grantului 376 finanţat de Banca Mondială şi Guvernul României. Mulţumesc fondatorilor Internet-ului şi directorului grantului cu pricina - care, întâmplător, este o persoană pe care o cunosc binişor.

Autorul Iaşi,
aprilie 2001

CONŢINUTUL LUCRĂRII

Cartea începe destul de anost, cu inevitabila terorie legată de bazele de date: cum au apărut, la ce sunt folositoare, care sunt componentele unui sistem de lucru cu bazele de date, ce este un sistem de gestiune a bazelor de date (SGBD) şi care îi sunt modulele principale etc. Pentru a mai atenua din durerile „teoretice", a fost inserată şi o istorie romanţată a SGBD-urilor relaţionale. Aceasta, de fapt, pentru a pregăti un nou calup teoretic, cel al noţiunilor modelului relaţional şi al restricţiilor ce pot fi definite pentru o bază de date. Din păcate, aceste paragrafe chiar trebuie citite, deoarece sintagmele explicate aici sunt folosite în capitolele următoare. Spre finalul primului capitol este prezentată în detaliu baza de date „martor" a lucrării, cea pe care vom opera comenzile SQL.

Al doilea capitol este jumătate teoretic şi restul practic. Teoretic, deoarece prezintă coordonatele esenţiale ale algebrei relaţionale, limbai curat... teoretic ce fundamentează SQL. Sunt prezentaţi *in extenso* operatorii ce permit extragerea informaţiilor dintr-o bază de date, cei asamblişti - reuniunea, intersecţia, diferenţa şi produsul cartezian - şi cei relaţionali - selecţia, proiecţia, joncţiunea (de fapt joncţiunile, că sunt mai multe tipuri) şi diviziunea. Aerul practic se datorează exemplelor care însoţesc fiecare operator prezentat, exemple ce utilizează baza de date descrisă în capitolul 1.

Al treilea capitol este şi cel care intră - e adevărat, nu chiar frontal - în problematica SQL. Nu chiar frontal, deoarece începe, aproape inevitabil pentru o carte scrisă de un universitar, cu o scurtă istorie a limbajului. Cu acest prilej sunt conturate şi direcţiile de evoluţie ale SQL. După enumerarea principalelor tipuri de date gestionabile prin SQL, discuţia se concentrează asupra modalităţilor de creare a tabelelor şi mai ales de declarare (cu greu am reuşit să evit cuvântul *implementare*) a restricţiilor unei baze de date. Formatul general din stardardul SQL-92 este permanent însoţit de variantele operaţionale în DB2, Oracle şi Visual FoxPro, lucru ce se va repeta în capitolele ce vor urma.

Apropo, de ce DB2, Oracle şi Visual FoxPro şi nu alte produse? în primul rând, pentru că sunt singurele dialecte SQL pe care le ştiu (cât de cât). în celelalte rânduri, pentru că:

- DB2 este produsul din care primele două standarde SQL au împrumutat cel mai mult; de altminteri, IBM este probabil firma-cheie atât în ceea ce priveşte apariţia modelului relaţional, cât şi SQL.
- Oracle deţine, de ani buni, cea mai mare cotă din piaţa bazelor de date şi nu sunt semne vizibile că ar trece curând pe locul doi.
- Visual FoxPro este primul SGBD „democrat" din ţara noastră - şi aceasta nu ţine atât de faptul că a apărut şi a fost utilizat după 1990, cât mai ales de aria sa de utilizare în ţara noastră, incomparabil mai mare decât pe plan mondial. Dacă, aşa cum spuneau cei din Şcoala Ardeleană, *de la Rîm ne tragem,* se poate spune că şi cel puţin o generaţie de dezvoltatori de aplicaţii cu baze de date din FoxPro se trage... Este drept că VFP nu stă chiar pe roze, ca evoluţie mondială a numărului de dezvoltatori ce-1 utilizează, dar mai este, pentru multe firme mici şi mijlocii, o soluţie ce nu trebuie ridiculizată.

Iar pe de altă parte, n-am reuşit să rezist tentaţiei de a alătura (numai într-o carte) trei produse ale unor firme (IBM, Oracle şi Microsoft) între care relaţiile sunt cordiale o dată sau de două ori pe deceniu.

Una din criticile pe care mi le asum este că nu am prezentat nici un dialect din categoria SGBD-rilor gratuite, astăzi din ce în ce mai populare: PostgreSQL, MySQL, Interbase etc. Poate în viitoarea ediţie a cărţii, după ce prezenta se va fi epuizat din anticariate şi de la buchinişti (vă daţi seama că lucrul acesta nu o să se petreacă prea curând, chiar dacă tirajul ar fi foarte mic).

Capitolul 4 este unul de introducere în interogările SQL, de prezentare a principalelor clauze ale frazei SELECT. în parte, exemplele sunt cele din algebra relaţională, dar discuţia depăşeşte cu mult cadrul celor opt operatori din capitolul 2. Astfel, sunt discutate: coloanele calculate (împreună cu opţiuni de concatenare şi operaţii cu date calendaristice), ordonarea liniilor în rezultatul unei interogări operatori pentru lucrul cu intervale (BETWEEN), liste (IN) şi şabloane (LIKE), sinonime locale şi autojoncţiune, funcţii de agregare (COUNT, SUM, AVG, MIN, MAX), precum şi clauzele de grupare (GROUP BY şi HAVING). ...

Capitolul 5 se apleacă asupra câtorva „delicatese" SQL: valori nule, joncţiune externă, structuri alternative generalizate (CASE), subconsultări (operatorul IN), operatorii ALL, SOME şi ANY şi utilizarea subconsultărilor în clauza HAVING a unei fraze SELECT. Soluţiile formulate pun în evidenţă avantajele şi limitele fiecăruia dintre cele trei produse - DB2, Oracle şi Visual FoxPro, în finalul capitolului sunt prezentate câteva variante de refugiu în SQL, în sensul că, atunci când dialectul produsului nu permite anumite elemente de fineţe, se poate recurge la varianta neortodoxă de a salva rezultatele consultărilor în tabele virtuale sau temporare şi folosirea acestora ca argumente în alte fraze SELECT, altfel spus, secvenţializarea interogărilor. Un bonus al paragrafului îl constituie prezentarea expresiilor-tabelă din DB2. _

După cum sugerează şi titulatura, capitolul 6 ridică frazele SELECT pe „cele mai înalte culmi ale SQL. Primul paragraf este dedicat uneia dintre cele mai dificile probleme în formularea consultărilor, corelarea simplă şi, mai ales, cea dublă. Prezentarea mecanismului corelărilor este însoţită de exemple ce pun în valoare utilitatea operatorului EXISTS. Al doilea paragraf este mult mai puţin încordat, evocând o facilitate preferată de mulţi dintre cei ce lucrează în SQL - includerea frazelor SELECT în clauza FROM a unei consultări/subconsultări, opţiune ce furnizează soluţii pentru multe probleme informaţionale aparent insolubile. _ . .

,

Reprezentarea structurilor ierarhice (arborescente) sunt tratate lapidar într-un paragraf special, împreună cu o serie de facilităţi Oracle în acest sens (clauzele CONNECT BY şi START WITH). Paragraful final al capitolului prezintă câteva ingrediente ale reţetei de actualizare a tabelelor prin comenzi UPDATE ce utilizează subconsultări corelate.

Singurul capitol care se apropie, timid, de cel mai recent standard SQL, SQL-99, este cel de-al şaptelea. Apropierea nu vizează, aşa cum probabil s-ar fi cuvenit, orientarea pe obiecte (în principal tipurile de date abstracte - ADT), ci un domeniu mult mai pragmatic, cel al analizei multidimensionale a datelor, domeniu consacrat în literatura de specialitate drept OLAP *(On Line Analytical Processing).* Nucleul OLAP din SQL-99: ROLLUP, CUBE, GROUPING, GROUPING SETS, RANK, ROW NUMBER etc. - este completat de câteva funcţii specifice Oracle 8i2: PERCENT_RANK, FIRST VALUE, LAST_VALUE, PERCENT_RANK, LAG şi LEAD. ^

Ultimul capitol, al optulea, dezvoltă câteva dintre elementele prezentate în precedentele capitole, cu privire la obiecte componente ale schemei unei baze de date şi intră în câteva detalii legate de extensiile

procedurale ale SQL în două dintre cele trei SGBD-uri, VFP şi Oracle. In completarea opţiunilor de creare a tabelelor prin combinaţia CREATE TABLE / SELECT şi declararea restricţiilor CHECK folosind subconsultări, sunt discutate modalităţile de creare şi actualizare a tabelelor virtuale.

Dintre extensiile procedurale ale SQL, câteva zeci de pagini sunt cheltuite cu procedurile şi funcţiile stocate în VFP şi Oracle, precum şi un tip special de proceduri stocate - declanşatoarele.

Vă urez lectură plăcută şi vise frumoase...

Capitolul 1 NOŢIUNI ALE MODELULUI RELAŢIONAL

’ 9

Modelul relaţional de organizare a datelor s-a conturat în două articole publicate în 1969 şi 1970 de către E.F. Codd, matematician la centrul de cercetări din San Jose (California) al firmei IBM[1]. în acel moment, tehnologia bazelor de date era centrată pe modelele *ierarhic* şi *reţea,* modele ce depind într-o mai mare măsură de organizarea internă a datelor pe suportul de stocare. Codd a propus o structură de date tabelară, independentă de tipul de echipamente şi software de sistem pe care este implementată, structură „înzestrată” cu o serie de operatori pentru extragerea datelor. Deşi puternic matematizat, modelul relaţional este relativ uşor de înţeles.

Dar înainte de a intra în subiect, vom cheltui câteva pagini cu teoria referitoare la noţiunile: fişiere de date, baze şi bănci de date, sisteme de gestiune a bazelor de date.

1.1. Baze de date şi sisteme de gestiune a bazelor de date

Conceptul de „bază de date” a apărut în a doua parte a anilor ’60. La momentul respectiv, în sistemele informatice ale organizaţiilor (atâtea câte erau - vorbesc de sisteme informatice), informaţiile erau organizate în fişiere de date (secvenţiale, indexate etc.), create cu ajutorul unor programe scrise în limbaje din a IlI-a generaţie: COBOL, Fortran etc. (figura 1.1).

Specific acestui mod de lucru, cunoscut ca *file-based* sau *flat files* (fişiere independente), este faptul că fiecare dată (Datai, Data2,... Datan) este descrisă (nume, tip, lungime), autonom, în toate fişierele în care apare. Mai mult, descrierea fiecărui fişier de date (câmpurile care-1 alcătuiesc, tipul şi lungimea fiecăruia, modul de organizare - sercvenţial, indexat, relativ etc.) este obligatorie în toate programele care îl „citesc” sau modifică. între FIŞIER/, FIŞIER2,... *FIŞIERn* nu există nici o relaţie definită explicit.

Spre exemplu, Data2 este prezentă în două fişiere de date, FIŞIER1 şi FIŞIER2. Dacă, prin program, se modifică formatul sau valoarea acesteia în FIŞIER 1, modificarea nu se face automat şi în FIŞIER2; prin urmare, o aceeaşi dată, Data2, va prezenta două valori diferite în cele două fişiere, iar necazurile bat la uşă: informaţiile furnizate de sistemul informatic sunt redundante şi prezintă un mare risc de pierdere a coerenţei.

---

1 Extrase din lucrarea [Codd70] se găsesc la adresa http://www.acm.org/classics/nov95/. De asemenea, o excelentă analiză a articolelor lui Codd este în [Date98],

Se poate proiecta un mecanism de menţinere a integrităţii datelor, astfel încât actualizarea unei date într-un fişier să atragă automat actualizarea tuturor fişierelor de date în care apare aceasta, însă, în sistemele mari, care gestionează volume uriaşe de informaţii, implementarea unui asemenea mecanism este extrem de complexă şi costisitoare. In plus, fişierele de date sunt uneori proiectate şi implementate la distanţe mari în timp, în formate diferite: de exemplu, FIŞIER1 este posibil să fi fost creat cu ajutorul limbajului COBOL, FIŞ1ER2 în FORTRAN, iar FIŞIER3 în BASIC. în asemenea condiţii, punerea în operă a mecanismului de menţinere a integrităţii devine o utopie.

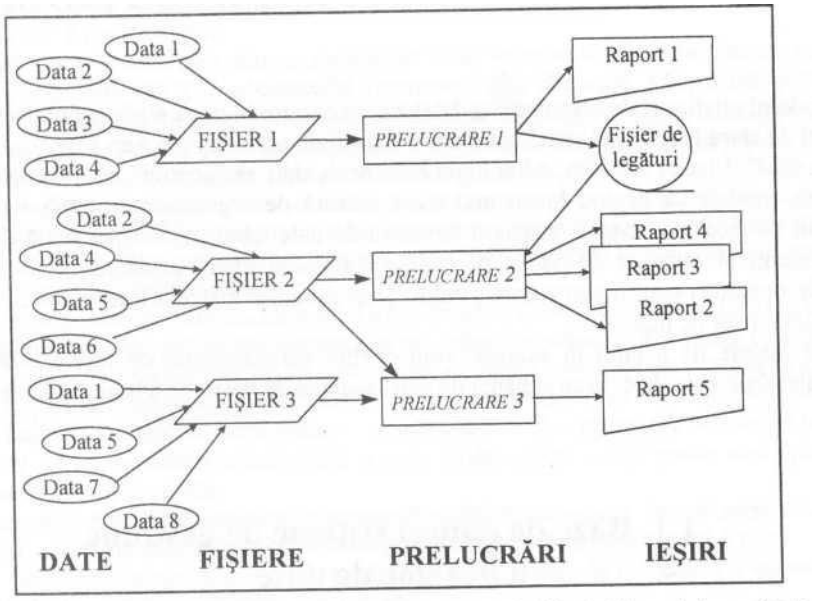


Figura 1.1. Sistem informatic bazat pe organizarea datelor în fişiere independente

Chiar numai şi din cele prezentate mai sus, se pot desprinde câteva dezavantaje ale organizării datelor după modelul *file-based* :

» Redundanţa şi inconsistenţa datelor: o aceeaşi dată apare în mai multe fişiere; în aceste cazuri există riscul modificării acesteia într-un fişier fară a face modificările şi în toate celelalte fişiere.

- Dificultatea accesului. într-o întreprindere, o aceeaşi informaţie este exploatată de mai mulţi utilizatori. Spre exemplu, pentru departamentul care se ocupă cu gestiunea stocurilor, intrările de materiale trebuie ordonate pe magazii (depozite) şi repere, în timp ce pentru departamentul care se ocupă cu decontările cu partenerii de afaceri ai întreprinderii, intrările trebuie ordonate pe furnizori ai materialelor. Or, fişierele tradiţionale nu facilitează accesarea datelor după mai multe criterii, specifice diferiţilor utilizatori sau grupuri de utilizatori.
- Izolarea datelor: când datele sunt stocate în formate diferite, este dificil de scris programe care să realizeze accesarea datelor într-o manieră globală a tuturor celor implicate în derularea unei tranzacţii.

- Complexitatea deosebită a actualizărilor. O actualizare presupune adăugarea, modificarea sau ştergerea unor informaţii din fişiere. Cum prelucrările se desfăşoară în timp real, de la mai multe terminale (în mediile multi-utilizator), pot apărea situaţii conflictuale atunci când doi utilizatori doresc modificarea simultană a unei aceleaşi date. Rezolvarea acestui gen de conflicte presupune existenţa unui program-supervizor al prelucrărilor, care este greu de realizat cu o multitudine de fişiere, create la distanţă în timp şi în formate diferite.
- Problemele de securitate ţin de dificultatea creării unui mecanism care să protejeze pe deplin datele din fişiere de accesul neautorizat.
- Probleme legate de integritatea datelor. Informaţiile stocate în fişiere sunt supuse la numeroase restricţii semantice. Toate aceste restricţii alcătuiesc mecanismul de integritate a datelor, deosebit de complex în mediile de lucru multi-utilizator şi eterogene.
- Inabilitatea de a obţine răspunsuri rapide la probleme ad-hoc simple.
- Costul ridicat se datorează gradului mare de redundanţă a datelor, eforturilor deosebite ce trebuie depuse pentru interconectarea diferitelor tipuri de fişiere de date şi pentru asigurarea funcţionării sistemului în condiţiile respectării unui nivel minim de integritate şi securitate a informaţiilor.

® Inflexibilitatea faţă de schimbările ulterioare, ce sunt inerente oricărui sistem informaţional.

- Modelarea inadecvată a lumii reale. •

Sintagma *bază de date* apare pentru prima dată în titlul unei conferinţe organizate la Santa Monica (California) în 1964 de System Development Corporation[3]. Majoritatea lucrărilor consideră ca moment al consacrării termenului anul 1969, când CODASYL publică, în cadrul unei conferinţe dedicate limbajelor de gestiune a datelor, primul raport tehnic [CODASYL69] în care este prezentat conceptul de „bază de date". Faţă de modelul *flle- -based,* noutatea o constituie existenţa unui fişier de descriere globală a bazei, astfel încât să se poată asigura independenţa programelor faţă de date, după cum o arată şi figura 1.2.

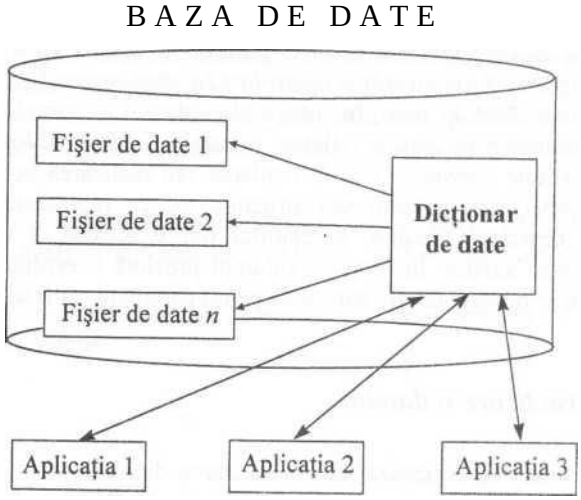


Figura 1.2. Schemă de principiu a unei baze de date

3. Conferinţa s-a intitulat *Development and Management of a Compnter-Centered Data Base.*

Avantajele organizării informaţiilor în baze de date decurg tocmai din existenţa acestui fişier de descriere globală a bazei, denumit, în general, *dicţionar de date* (denumit şi repertoar de date sau catalog de sistem). Extragerea şi modificarea datelor - altfel spus, lucrul cu fişierele de date - se derulează exclusiv prin intermediul dicţionarului în care se găsesc informaţii privitoare la structura datelor şi restricţiile îndeplinite de acestea. Iată câteva dintre avantaje:

- Un grad redus de redundanţă a datelor.
- Evitarea, în mai mare măsură, a inconsistenţei datelor.
- Facilitarea partajării informaţiilor între toţi utilizatorii acestora din cadrul organizaţiei.
- Suport pentru standardizare.
- Implementarea unor mecanisme ameliorate privind asigurarea securităţii informaţiilor.
- îmbunătăţirea integrităţii datelor.
- Un mai bun suport pentru rezolvarea conflictelor ce apar la încercările de modificare simultană a unei aceleiaşi date (de către doi sau mai mulţi utilizatori).
- » Structurile de date sunt mai aproape de realitate şi mai uşor de manipulat.
- Este permisă legătura cu diverse limbaje-gazdă.
- întreprinderea poate fi abordată global, luându-se în considerare şi interacţiunile dintre compartimente (producţie, marketing, personal, finanţe, contabilitate).
- Datele fiind separate de programele de consultare şi actualizare a lor, procesul de dezvoltare a aplicaţiilor-program este sensibil ameliorat, efortul de scriere a programelor (codarea) diminuându-se considerabil.
- Sistemele informatice ce utilizează baze de date sunt mai flexibile, reflectă mai bine specificul firmei, fiind adaptabile la modificările ulterioare ale mediului economic.

O bază de date (BD) reprezintă un ansamblu structurat de fişiere care grupează datele prelucrate în aplicaţiile informatice ale unei persoane, grup de persoane, întreprinderi, instituţii etc. Formal, *BD poate fi definită ca o colecţie de date aflate în interdependenţă, împreună cu descrierea datelor şi a relaţiilor dintre ele,* iar după G.D. Everest, *o BD reprezintă o colecţie de date utilizată într-o organizaţie, colecţie care este automatizată, partajată, definită riguros (formalizată) şi controlată la nivel central .*

Bazele de date evoluează în timp, în funcţie de volumul şi complexitatea proceselor, fenomenelor şi operaţiunilor pe care le reflectă. Ansamblul informaţiilor stocate în bază la un moment dat constituie *conţinutul* sau instanţierea sau realizarea acesteia. Organizarea bazei de date se reflectă în *schema* sau structura sa, ce reprezintă un ansamblu de instrumente pentru descrierea datelor, a relaţiilor dintre acestea, a semanticii lor şi a restricţiilor !a care sunt supuse. în timp ce volumul prezintă o evoluţie spectaculoasă în timp, schema unei baze rămâne relativ constantă pe tot parcursul utilizării acesteia.

Niveluri de abstractizare a datelor

într-un sistem informatic ce utilizează BD, organizarea datelor poate fi analizată din mai multe puncte de vedere şi pe diferite paliere. De obicei, abordarea se face pe trei niveluri. *fizic* sau intern, *conceptual* sau global şi *extern.*

Nivelul Fizic (sau **intern).** Reprezintă modalitatea efectivă în care datele sunt „scrise" pe suportul de stocare - disc magnetic, disc optic, bandă magnetică etc. Structura datelor este descrisă foarte detaliat, fiind accesibilă numai specialiştilor (ingineri de sistem, programatori în limbaje de asamblare sau alte limbaje apropiate de „maşină"). Cele două părţi principale ale bazei la acest nivel sunt: (1) un set de programe care interacţionează cu sistemul de operare pentru îmbunătăţirea managementului bazei

de date şi (2) fişierele stocate în memoria externă a calculatorului. Fişierele ce conţin datele propriu-zise sunt alcătuite din articole sau înregistrări cu format comun. La acest nivel, structura BD se concretizează în *schema internă.*

Nivelul conceptual (sau **global).** Este nivelul imediat superior celui fizic, datele fiind privite prin prisma semanticii lor; interesează conţinutul lor efectiv, precum şi relaţiile care le leagă de alte date. Reprezintă primul nivel de abstractizare a lumii reale observate. Obiectivul acestui nivel îl constituie modelarea realităţii considerate, asigurându-se independenţa bazei faţă de orice restricţie tehnologică sau echipament anume. întreaga bază este descrisă prin intermediul unui număr restrâns de structuri. Toţi utilizatorii îşi exprimă nevoile de date la nivel conceptual, prezentându-le administratorului bazei de date, acesta fiind cel care are o viziune globală necesară satisfacerii tuturor cerinţelor informaţionale. La acest nivel, structura BD se concretizează în *schema conceptuală.*

Nivelul extern. Este ultimul nivel de abstractizare la care poate fi descrisă o bază de date. Structurile de la nivelul conceptual sunt relativ simple, însă volumul lor poate fi deconcertant. Iar dacă la nivel conceptual baza de date este abordată în ansamblul ei, în practică, un utilizator sau un grup de utilizatori lucrează numai cu o porţiune specifică a bazei, în funcţie de departamentul în care îşi desfăşoară activitatea şi de atribuţii. Simplificarea interacţiunii utilizatori-bază, precum şi creşterea securităţii bazei sunt deziderate ale unui nivel superior de abstractizare, care este nivelul extern. Astfel, structura BD se prezintă sub diferite machete, cunoscute uneori şi *ca sub-scheme,* scheme externe sau imagini, în funcţie de nevoile fiecărui utilizator sau grup de utilizatori.

Luând în considerare cele trei niveluri de abstractizare, schematizarea unui sistem de lucru cu o bază de date se poate face ca în figura 1.3[2].

Accesul utilizatorului la informaţiile din bază este posibil numai prin intermediul sistemului de gestiune a bazei de date. în general, interfaţa utilizator-SGBD se poate realiza în două moduri:

- Printr-un mecanism de apel (cuvânt-cheie, ca de exemplu CALL) inserat în programele scrise într-un limbaj „tradiţional" (C, COBOL etc.), acesta fiind cazul SGBD-urilor cu limbaj-gazdâ.
- Prin comenzi speciale utilizate autonom (în afara aplicaţiilor-program), în cazul SGBD-urilor autonome.

2 Prelucrare a schemei din [Date86], p. 33.

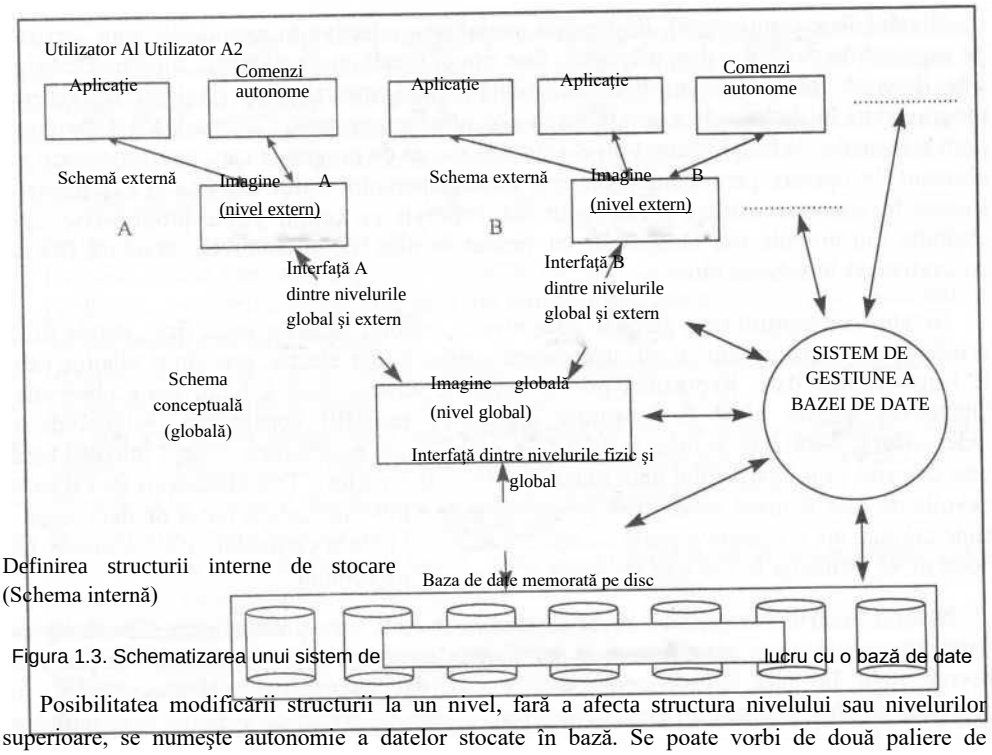


Figura 1.3. Schematizarea unui sistem de lucru cu o bază de date

Posibilitatea modificării structurii la un nivel, fără a afecta structura nivelului sau nivelurilor superioare, se numeşte autonomie a datelor stocate în bază. Se poate vorbi de două paliere de autonomie.

Autonomia fizică reprezintă posibilitatea modificării arhitecturii bazei la nivel intern, fară ca aceasta să necesite schimbarea schemei conceptuale şi rescrierea programelor pentru exploatarea bazei de date. Asemenea modificări sunt necesare uneori pentru ameliorarea performanţelor de lucru (viteză de acces, mărimea fişierelor etc.). Tot autonomia fizică este cea care asigură portarea bazei de date de pe un sistem de calcul pe altul fară modificarea schemei conceptuale şi a programelor.

Autonomia logică presupune posibilitatea modificării schemei conceptuale a bazei (modificare datorată necesităţii rezolvării unor noi cerinţe informaţionale) fără a rescrie programele de exploatare. Autonomia logică a datelor este mai greu de^ realizat decât autonomia fizică, deoarece programele de exploatare sunt dependente, în foarte mare măsură, de structura logică a datelor pe care le consultă şi actualizează, în ciuda existenţei dicţionarului de date. Fireşte, un element important îl reprezintă şi anvergura modificării schemei conceptuale.

Sisteme de gestiune a bazelor de date

Apărute în anii ’60, Sistemele de Gestiune a Bazelor de Date (prescurtat SGBD-uri) reprezintă *un ansamblu de programe ce permit utilizatorilor să interacţioneze cu o bază de date, în vederea creării, actualizării şi interogării acesteia.* SGBD-ul este cel care asigură şi supervizează: introducerea de informaţii în baza de date; actualizarea şi extragerea datelor din bază; autorizarea şi controlul accesului la date; păstrarea independenţei dintre structura bazei şi programe[3].

Obiectivul esenţial al unui SGBD este furnizarea unui mediu eficient, adaptat utilizatorilor care doresc să consulte sau să actualizeze informaţiile conţinute în bază. Bazele de date sunt concepute pentru a prelucra un volum mare de informaţii. Gestiunea acestora impune nu numai o structurare riguroasă a datelor, dar şi o raţionalizare a procedurilor de acces şi prelucrare.

Principalele funcţiuni ale unui SGBD vizează:

- descrierea ansamblului de date la nivelurile fizic şi conceptual;
- crearea (iniţializarea) şi exploatarea (consultarea şi actualizarea) bazei de date;
- » controlul integrităţii bazei;
- confidenţialitatea informaţiilor conţinute în bază;
- accesul simultan al mai multor utilizatori la informaţii;
- ® securitatea în funcţionare;
- furnizarea unui set de comenzi şi instrucţiuni necesare atât utilizatorilor pentru consultarea directă a bazei, prin intermediul unui limbaj de manipulare, cât şi programatorilor, pentru redactarea programelor de lucru cu baza de date;
- revizia şi restructurarea bazei;
- ® monitorizarea performanţelor.

Un SGBD prezintă, în general, următoarele module[4]:

© Gestionarul fişierelor, care se ocupă cu afectarea spaţiilor de memorie pe disc şi cu structurile fizice de date care servesc la reprezentarea informaţiilor pe suport.

© Gestionarul bazei de date face legătura datelor „fizice” din bază cu aplicaţiile-program de consultare şi actualizare.

© Procesorul de consultare „traduce” instrucţiunile limbajului de consultare în instrucţiuni elementare, „inteligibile” pentru gestionarul bazei. Mai mult, acesta optimizează consultarea, pentru a obţine rezultatele într-un timp cât mai scurt.

© Modulele DML (*Data Manipulation Language*) realizează conversia instrucţiunilor limbajului de manipulare a datelor (DML) inserate într-un program de aplicaţie în proceduri curente ale limbajului-gazdă, interacţionând cu procesorul de consultare în vederea producerii secvenţelor de cod adecvate.

© Modulele Limbajului de Definire a Datelor — DDL (*Data Definition Language*) „traduc” (prin compilare sau interpretare) şi execută instrucţiunile DDL, obţinându-se ansamblul de tabele ce reprezintă metadatele stocate în dicţionarul de date.

3 [G. Dodescu ş.a. 87], p. 511.
4 [Korth&Silberschatz88], pp. 17-18.

Aceste cinci module interacţionează cu o serie de componente fizice ale bazei.

Fişierele de date reprezintă suportul propriu-zis al bazei.

Dicţionarul de date înmagazinează informaţii relative la structura bazei, fiind solicitat în toate operaţiunile de consultare şi actualizare.

Indecşi, într-un număr suficient pentru mărirea vitezei de acces la date.

Sintetic, structura unui sistem de lucru cu o bază de date este prezentată în figura 1.4 .

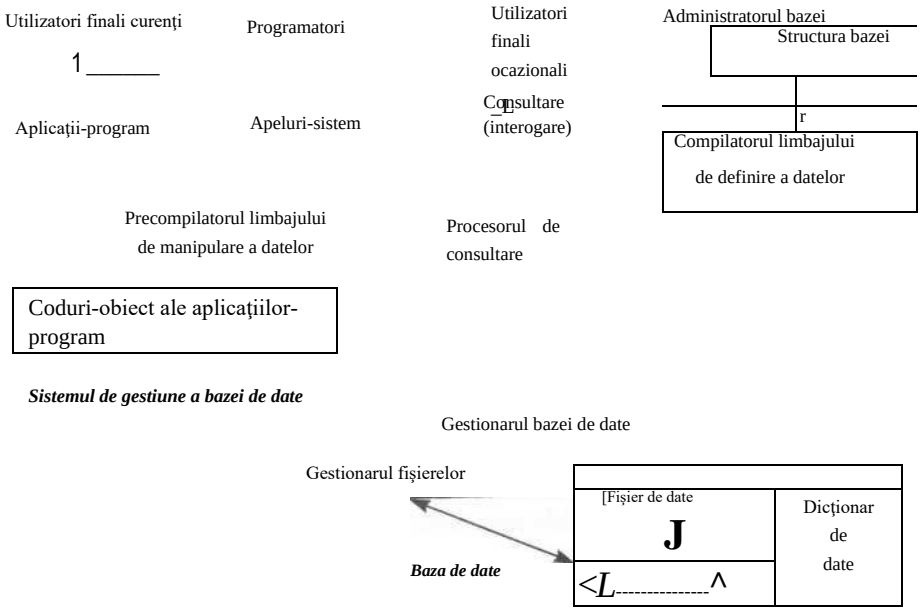


Figura 1.4. Structura unui sistem de lucru cu o bază de date

Limbaje de definire a datelor

Arhitectura unei baze de date este specificată printr-o serie de definiţii redactate sub formă de instrucţiuni scrise într-un anumit limbaj - limbajul d6 definire a datelor - DDL *(Pata Defmition Language).* Execuţia acestor definiţii se materializează într-un ansamblu e tabele care sunt memorate într-un fişier special, denumit dicţionar (sau repertoar) de date.

Un dicţionar de date conţine deci metadate, adică date relative la alte date, fund consultat înaintea oricărei citiri sau modificări a datelor din bază.

Principalele funcţii ale DDL sunt:

. Descrierea logică a bazei de date şi sub-schemelor proprii fiecărui grup de utilizatori.

- Identificarea schemei, sub-schemelor şi diverselor agregări de date.

- Specificarea fişierelor de date şi a legăturilor logice dintre acestea. Pe baza acestor specificaţii se poate realiza accesul la date chiar şi în condiţiile coexistenţei mai multor modele de organizare într-o aceeaşi BD.
- Definirea restricţiilor semantice la care sunt supuse datele. Acestea reprezintă ansamblul valorilor permise ale fiecărei date, eventual formula de calcul a unei date pe baza valorilor altor date. Respectarea acestor restricţii asigură coerenţa bazei.
- Definirea cheilor de acces rapid şi a cheilor confidenţiale (parolelor).
- Definirea metodelor de exploatare a fişierelor ce vor fi utilizate în aplicaţii pentru selectarea înregistrărilor.
- Definirea procedurilor speciale de criptare, în vederea generării cheilor de acces.
- Definirea modului de indexare sau de localizare a entităţilor.
- Determinarea tipului unei date, în sensul de dată de bază sau derivată (calculată printr-o expresie, pe baza valorilor altor date).

Dată fiind complexitatea structurilor de memorare şi metodelor de acces la acestea, la nivel elementar fişierele de date şi dicţionarul de date sunt accesibile numai specialiştilor. In schimb, pentru descrierea schemei BD, marea majoritate a limbajelor de interogare prezintă comenzi adecvate, ce pot fi utilizate şi de ne-informaticieni.

Limbaje de manipulare a datelor

Prin manipularea datelor se înţelege efectuarea uneia dintre următoarele operaţiuni:

- extragerea unor date din bază (consultare);
- scrierea de noi date în bază (adăugare);
- ştergerea datelor perimate sau eronate (uneori chiar şi a celor corecte);
- modificarea valorii unor date.

Un limbaj de manipulare a datelor (DML — *Data Manipulation Language*) este utilizat pentru a prelucra datele în funcţie de structura lor. La nivel fizic, prin DML se vizează identificarea şi implementarea unor algoritmi performanţi de acces la date, în timp ce la nivel extern DML trebuie să faciliteze dialogul utilizatorului cu baza în vederea obţinerii informaţiilor dorite.

In mod curent, termenul *consultare* sau *interogare* desemnează acţiunea de căutare şi identificare a datelor necesare, dintre cele aflate în bază. Ansamblul instrucţiunilor DML pentru căutarea şi identificarea datelor constituie limbajul de consultare.

Gestionarul bazei

O bază de date consumă un volum considerabil de memorie, astfel încât poate fi stocată numai pe disc, memoria internă (RAM) a calculatorului fiind insuficientă (şi volatilă). Programele de exploatare fac frecvent transferuri de date între memoria internă şi disc. Deoarece viteza de transfer este mult mai mică, prin comparaţie cu viteza de lucru a procesorului, găsirea unor soluţii, la nivel fizic, pentru transferul rapid al datelor prezintă o importanţă deosebită.

Pe de altă parte, obiectivul esenţial al bazelor de date este de a facilita exploatarea datelor, în vederea obţinerii de către utilizatori a informaţiilor necesare. Acest obiectiv priveşte nivelul extern, utilizatorul nefiind realmente interesat de modul în care informaţia se găseşte fizic pe suportul de înregistrare.

Rezolvarea acestor probleme cade sub incidenţa gestionarului bazei de date, care este un modul de programe ce realizează interfaţa dintre datele interne (de pe suport) conţinute în bază şi programele (sau comenzile) de consultare şi actualizare; principalele sale sarcini pot fi grupate astfel:

- Interacţiunea cu gestionarul de fişiere. Datele brute sunt stocate pe disc prin intermediul sistemului de gestiune a fişierelor, care este o componentă a sistemului de operare al

calculatorului. Gestionarul BD traduce instrucţiunile diverselor DML în instrucţiuni- -sistem, la nivel elementar, fiind „responsabil" şi de buna desfăşurare a operaţiilor de scriere/citire a datelor în/din bază.

- Validitatea (corectitudinea) datelor. Datele stocate trebuie să satisfacă anumite restricţii de integritate, specificate de administratorul bazei. După implementarea mecanismului de integritate, gestionarul verifică dacă toate actualizările se derulează cu respectarea restricţiilor.
- Securitatea datelor. Accesul la date trebuie să fie selectiv, gestionarul bazei fiind cel care asigură respectarea drepturilor de acces ale fiecărui utilizator.
- Salvare şi restaurare. Şi sistemele informatice sunt supuse accidentelor: distrugerea suprafeţei magnetice, întreruperi în furnizarea energiei electrice, erori ale programelor etc. sunt inerente chiar şi în ţări super-dezvoltate. în multe cazuri, datele ce erau în curs de prelucrare în momentul accidentului se alterează sau se pierd în totalitate. Gestionarul are dificila misiune de a detecta la timp avariile şi de a restaura datele pierdute în forma şi conţinutul dinaintea accidentului. Aceasta se realizează prin intermediul unor programe speciale de salvare-restaurare.
- Prelucrări simultane (concurente). întrucât în acelaşi timp pot lucra cu baza doi sau mai mulţi utilizatori, trebuie asigurată coerenţa datelor prin afectarea unor niveluri de prioritate la nivel de operaţii şi utilizatori.

Administratorul bazei de date

Din cele prezentate până acum reiese clar că un sistem de gestiune a bazei de date reprezintă un ansamblu de programe ce asigură controlul complet al datelor, dar şi al aplicaţiilor care le exploatează. Administratorul unei baze de date este persoana responsabilă de sistem în ansamblul său. Rolul acestuia este determinant în câteva activităţi foarte importante.

- Definirea arhitecturii bazei de date se realizează prin scrierea definiţiilor care vor fi transformate de compilatorul DDL într-un ansamblu de tabele stocate permanent în dicţionarul de date.
- Definirea modalităţilor în care va fi structurată memoria externă şi a metodelor de acces la date. Acestea devin operaţionale prin redactarea unor specificaţii scrise ca instrucţiuni ale limbajului de definire (descriere) a datelor.
- Modificarea arhitecturii şi organizării fizice a bazei de date poate fi realizată prin intermediul instrucţiunilor DDL, obţinându-se astfel actualizările corespondente ale dicţionarului de date.
- Autorizarea accesului la date se acordă fiecărui utilizator al bazei de date, administratorul fiind cel care decide asupra datelor ce pot fi consultate şi actualizate de fiecare utilizator sau grup de utilizatori.
- Specificarea restricţiilor de integritate. Aceste restricţii sunt stocate pe disc şi consultate de gestionarul bazei Ia fiecare actualizare.

în plus, administratorul bazei de date este cel care: asigură legătura cu utilizatorii; defineşte procedurile de verificare a drepturilor de acces şi a procedurilor de validare a integrităţii datelor; defineşte strategia de salvare (înregistrarea copiilor de siguranţă)/restaurare a bazei; monitorizează performanţele bazei şi o adaptează la modificările ulterioare ale sistemului informaţional.

După cum se poate observa, sarcina unui administrator este deosebit de complexă, iar rolul său este primordial în exploatarea bazei de date. De aceea, raportul ANSI/SPARC 1975 delimitează trei categorii de administratori:

- Administratorul întreprinderii, responsabil cu exploatarea şi identificarea globală a aplicaţiilor prezente şi viitoare utilizate în organizaţia respectivă;
- Administratorul aplicaţiilor, responsabil cu dezvoltarea sub-schemelor externe pentru aplicaţii şi utilizatori;

- Administratorul datelor precizează necesarul şi disponibilitatea datelor la nivelul schemei interne.

Utilizatorii bazelor de date

Cu riscul de a simplifica lucrurile excesiv, ne putem opri numai la patru categorii de utilizatori ce pot interacţiona cu sistemul:

Programatorii de aplicaţii. Sunt informaticienii care interacţionează cu baza de date prin intermediul instrucţiunilor DML integrate în programe scrise într-un limbaj-gazdă (Cobol, Pascal, C, Visual Basic, Java) sau în limbajul SGBD-lui respectiv.

Utilizatorii ocazionali. Aceştia sunt persoane care interacţionează cu sistemul nu prin intermediul programelor de aplicaţii, ci printr-un limbaj de consultare specific bazei. Fiecare interogare este transmisă unui procesor de consultare având rolul de a prelucra instrucţiunile DML şi a le face „inteligibile" pentru gestionarul bazei.

Utilizatorii curenţi sunt persoane care efectuează operaţii de rutină asupra bazei de date, utilizând aplicaţiile disponibile.

Utilizatorii specializaţi reprezintă un ansamblu de specialişti care dezvoltă aplicaţii- -program ce lucrează cu baze de date, dar în afara cadrului general al SGBD-urilor. Aici sunt incluşi specialiştii în sisteme expert, proiectare asistată de calculator, modelare etc.

1.2. Un pic de istorie recentă şi relaţională

Codd a definit modelul relaţional al datelor printr-o serie de *structuri* de date (relaţii alcătuite din tupluri), *operaţii* aplicate asupra structurilor de date (selecţie, proiecţie, joncţiune etc.) şi *reguli de integritate* care să asigure consistenţa datelor (chei primare, restricţii referenţiale ş.a.).

Modelarea realităţii se concretizează în *tabele de valori* numite relaţii, avându-se în vedere că:

- relaţie are un *nume*
- coloană reprezintă un *atribut',*
- linie reprezintă un *n-uplet ituplu)* de valori ale celor *n* atribute din relaţie;
- ordinea liniilor şi coloanelor în cadrul tabelei nu este relevantă pentru conţinutul informaţional.

Fiecare linie a tabelei reprezintă o entitate sau un fapt al realităţii, în timp ce o coloană reprezintă o proprietate a acestei entităţi sau fapt. Introducând un plus de rigoare, se cuvine de remarcat că entitatea poate fi nu numai un obiect concret/proces (o persoană, un lucru oarecare), dar şi o relaţie între obiecte (persoane, lucruri), cum ar fi: contractele de afaceri, căsătoriile (există totuşi câteva deosebiri între acestea şi precedentele), structura ierarhică a unei organizaţii etc. în plus, nu e întotdeauna evident care dintre informaţii sunt entităţi, care atribute şi care asociaţii (relaţii).

Dintre coloane, o parte are drept scop identificarea liniilor (*identificator* sau *cheie primara)-*, altele constituie suportul de referinţă către linii din alte tabele (*coloane de referinţă* sau *chei străine).*

Ansamblul valorilor stocate în tabele reprezintă *conţinutul* bazei de date, conţinut ce poate fi modificat prin operaţiuni de actualizare: introducerea unor linii (tupluri) noi, ştergerea unor linii, modificarea valorii unor atribute.

In figura 1.5[5] este prezentată o schemă incompletă de evoluţie a bazelor de date, din punctul de vedere al modelului intern de organizare a datelor. Prima etapă este reprezentată de sisteme care utilizează modelul *file-based* (pre-baze de date), a doua - modelul *reţea,* a treia - modelul *relaţional,*

5 Preluare din M. Sârbu - „Dincolo de relaţional?", *PC Report,* nr. 35 (august), 1995.

iar ultima, în curs de maturizare, este etapa SGBD-urilor *orientate pe obiecte.* Schemei i se poate reproşa absenţa unui model important, din care a derivat cel reţea, şi anume modelul *ierarhic.*

Faţă de modelele ierarhice şi reţea, modelul relaţional prezintă câteva avantaje[10]:

- propune structuri de date uşor de utilizat;
- ameliorează independenţa logică şi fizică;
- pune la dispoziţia utilizatorilor limbaje ne-procedurale;
- optimizează accesul la date;
- îmbunătăţeşte integritatea şi confidenţialitatea datelor;
- ia în calcul o largă varietate de aplicaţii;
- abordează metodologic definirea structurii bazei de date.

Modelului relaţional îi este asociată *teoria normalizării,* care are ca scop prevenirea comportamentului aberant al relaţiilor în momentul actualizării lor, eliminarea datelor redundante şi înlesnirea înţelegerii legăturilor semantice dintre date.

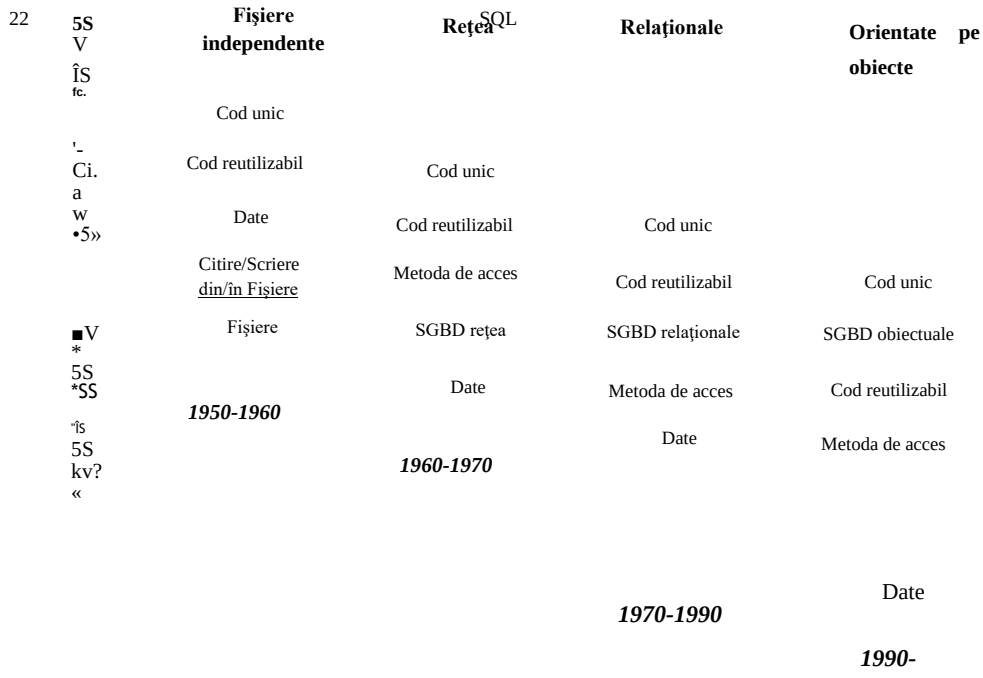


Figura 1.5. Evoluţia bazelor de date

Articolele lui Codd au avut un ecou semnificativ în lumea cercetătorilor. Firma IBM a iniţiat un proiect major care trebuia să se materializeze în implementarea unui sistem de gestiune a bazelor de date relaţionale, numit System/R. Proiectul s-a derulat în laboratorul Santa Teresa din San Jose, California. Prima fază — anii 1974 şi 1975 — s-a concretizat într-un prototip al SGBDR-ului, prototip care a fost rescris în anii 1976 şi 1977 pentru a permite interogarea simultană a mai multor tabele şi accesul multi-utilizator. IBM a distribuit produsul pe parcursul anilor 1978 şi 1979 la câţiva colaboratori ai săi pentru evaluări, dar în 1979 firma abandonează proiectul System/R, considerându-1 nefezabil". System/R a fost însă punctul de plecare pentru realizarea unuia dintre cele mai cunoscute SGBDR-uri, DB2.

Intre timp, în 1977, un grup de ingineri de la Menlo Park, California, a fondat compania Relaţional Software Inc., cu scopul declarat de a dezvolta un SGBD bazat pe limbajul SQL. Produsul, denumit Oracle, a fost lansat în 1979 şi a reprezentat primul SGBDR comercial.

Dincolo de faptul că Oracle precedă cu doi ani primul SGBDR comercial al firmei IBM, orientarea sa a fost extrem de bine gândită, rulând pe minicalculatoarele VAX (Digital Equipment Corporation), sensibil mai ieftine decât mainframe-urile IBM, şi, deci, cu un segment de piaţă mai mare. Astfel, succesul a fost vizibil. Astăzi, firma, redenumită Oracle Corporation, este liderul produselor software dedicate mediilor de lucru cu bazele de date.

Primul SGBDR comercializat de IBM a fost SQL/DS (DS — *Data System),* anunţat în 1981 şi comercializat în 1982. în 1983, IBM a lansat DB2, dedicat iniţial mainframe-urilor sale, dar care în timp a fost portat pe toate sistemele de operare importante.

Alte SGBD-uri din perioada de pionierat a relaţionalului au fost: INGRES (*Interactive Graphics and Retrieval System)* - al Universităţii Berkeley; QUERY BY EXAMPLE - Centrul de cercetări T.J. Watson; PRTV (*Peterlee Relationnel Test Vehicle)* - Centrul

Ştiinţific Peterlee (Marea Britanie), patronat de firma IBM; RDMS (*Relaţional Data Management System*) - General Motors; INFORMIX (denumită iniţial *Relaţional Data Systems*) etc. ...

Până la jumătatea anilor ’80, SGBD-urile relaţionale au stat în umbra celor ierarhice şi reţea. Motivul principal l-a constituit viteza mai mică, parametru esenţial în sistemele „confruntate” cu accesul simultan a sute sau chiar mii de utilizatori. în a doua parte a anilor ’80, performanţele SGBDR-urilor au fost net ameliorate. Câştigul de viteză, care s-a adăugat atuului principal, interogarea ad-hoc prin utilizarea limbajelor de generaţia a IV-a (SQL, QBE, Quel), a lărgit continuu segmentul ocupat de SGBDR-uri, acesta fiind estimat în 1997 la aproximativ 60% din totalul pieţei bazelor de date[6].

Practic, pentru toate platformele (mainframe-uri, minicalculatoare, staţii de lucru, PC-uri) şi sistemele de operare există o întreagă gamă de SGBDR-uri deosebit de performante, printre „protagonişti” numărându-se produsele firmelor: Oracle, IBM, Sybase, Informix, Microsoft etc.

O adevărată „democratizare” a SGBDR-urilor s-a produs o dată cu dezvoltarea explozivă a PC-urilor. Produse precum dBase, RBase, Clipper, FoxPro, Paradox, Access etc. s-au vândut (şi copiat ilegal) în milioane de exemplare. Destinate iniţial utilizatorilor de PC-uri cu un redus bagaj de cunoştinţe în domeniul bazelor de date, astăzi aceste produse sunt dotate cu seturi puternice de instrucţiuni şi funcţii şi prezintă o interfaţă de lucru deosebit de prietenoasă, atuuri ce conferă o mare dinamică acestui segment al pieţei, denumit şi cel al *SGBD-urilor micro.*

Exploatabile pe platformele de lucru MS-DOS/Windows, reprezintă zona „de jos” care vizează depotrivă ne-profesioniştii în ale informaticii, dar şi dezvoltatori de aplicaţii la scară medie. Totodată, în reţelele complexe (de întreprindere), SGBD-urile *micro* pot fi utilizate ca medii de dezvoltare pentru staţiile *client,* prin care se asigură accesul utilizatorilor la BD administrate printr-un SGBDR „profesional”.

începuturile acestei familii de produse-program se pierd în negura istoriei... anilor ’70. Un anume Wayne Ratcliff a elaborat un SGBD simplu, dedicat microcalculatoarelor. La acel moment, acestea erau echipate cu microprocesoare pe 8 biţi (Intel 8080, Zilog Z80, Motorola 6800 etc.), memorie internă de ordinul a zeci de kiloocteţi, unităţi de dischete de 256 Ko şi alte „facilităţi” care ar strânge în spate astăzi pe orice utilizator de PC dotat cu procesor Pentium II sau III. Produsul este preluat de firma Ashton-Tate (la care George Tate era director), care îl redenumeşte *dBase.* Versiunea *dBase II* a avut parte de o intrare triumfală în familia programelor pentru microcalculatoare. Paradoxal sau nu, nu a existat un dBase I, Tate considerând că lansarea produsului cu sufixul II ar sugera utilizatorului că produsul este deja matur. Şi aşa a fost... După *dBase II* urmează versiunea *III,* apoi *III plus,* care va cunoaşte gloria, fiind etalonul „de facto” pe piaţa nou-apărută a SGBD-urilor pentru PC-uri.

Visul frumos al firmei Ashton-Tate s-a diluat şi apoi spulberat o dată cu versiunea *dBase IV.* Intens mediatizată şi aşteptată destul de mult, prima versiune, *dBase IV 1.0,* a fost „presărată” cu erori atât de hilare, încât produsul a devenit rapid emblema deceniului opt şi începutului de deceniu 9 pentru programele făcute în pripă şi care prezintă multe erori sau, cum sugestiv sună în limba engleză, *quick and dirty* („în grabă şi murdar”). Este adevărat, „ştafeta” a fost preluată de alte produse (Corel Draw 5 este numai un exemplu), dar concurenţa e acerbă, chiar case mari (în primul rând Microsoft) fiind ţinta ironiilor datorate „bug”-urilor de care nu duc lipsă pachetele de programe pe care le produc.

Pentru Ashton-Tate, dBase IV 1.0 a fost începutul declinului. Datorită dificultăţilor financiare, în 1991 firma a fost preluată de Borland (astăzi Inprise), care n-a reuşit, nici ea, să-şi revină (financiar) din „şocul” Ashton-Tate.

Intre timp, au apărut şi câştigat poziţii avansate produsele competitoare. Unul dintre cele mai cunoscute este FoxPro. La mijlocul deceniului 8, la Seattle (SUA), firma Fox Software a atras atenţia prin produsul său principal, FoxBase, care era o „clonă” dBase. La scurt timp însă, FoxBase-uI a surclasat „maestrul” dBase Ia aproape toate capitolele: viteză, interactivitate, pluralitate de platforme

6 Preluare din *01 Reseaux,* nr. 39 (iunie), 1997, p. 107.

(încă din anii ’80 a fost disponibilă o versiune FoxBase pentru Macintosh) etc.

Ulterior FoxBase devine FoxPro, anul 1991 fiind anul în care marea majoritate a publicaţiilor de specialitate au „încoronat la categoria SGBD-uri pentru microcalculatoare, pe FoxPro 2.0, iar altele, precum *PC Magazine, InfoWorld, PC Computing, Byte,* l-au desemnat drept „cel mai bun produs-prograni al anului 1991”[lj]. Este şi motivul pentru care în 1992, firma Microsoft, care la acea dată nu avea un SGBD de forţă la această categorie, a înghiţit Fox Software şi, începând cu versiunea 2.5, FoxPro poartă sigla gigantului din Redmont (statul Washington). Unii spun că de aici începe şi declinul FoxPro.

Dintre ceilalţi concurenţi puternici ai dBase-ului amintim doar Paradox (Borland), RBase (Microrim), Clipper (Nantucket, companie fondată de doi ex-Ashton-Tate şi cumpărată ulterior de Computer Associates), dBFast (Computer Associates), Approach (Lotus, achiziţionată în 1995 de IBM), Access (Microsoft) etc.

Lumea *SGBD-urilor micro* a suferit schimbări majore atât în fizionomie, dar şi în modul efectiv de lucru, mai ales pentru cele din clasa XBase. Visual Objects (Computer Associates), Visual FoxPro, Visual dBase seamănă din ce în ce mai puţin cu ceea ce a fost altădată un produs XBase, fiind instrumente care înglobează tehnologii recente: client-server, replicare, programare orientată pe obiecte, dezvoltarea rapidă a aplicaţiilor (RAD) etc.

Este greu de făcut o ierarhizare, însă piaţa a consacrat la această categorie produsul Access al firmei Microsoft. Ajuns la versiunea 2000, inclus în suita Office cu acelaşi sufix, Access este ieftin, intuitiv şi bine integrat cu ceilalţi „pilieri”, Excel şi Word. Astfel încât aria sa de utilizare este superioară oricărui alt produs competitor.

Urcând la categoria superioară SGBDR-urilor micro, se cuvinte de amintit că piaţa a cochetat pe la mijocul anilor ’90 cu sintagma *servere de date pentru grupuri de lucru*[7]. Acest concept a fost pus în practică de firma Microsoft, prin comercializarea la un preţ foarte avantajos a produsului *SQL Server,* un SGBD ce permitea iniţial accesul simultan la baza de date a unui număr de 20-30 utilizatori. Fără a se ridica la pretenţiile unui SGBD Unix, SGBDR-urile pentru grupurile de lucru au fost declarate instrumentul ideal pentru firmele mici şi mijlocii, acolo unde Paradox, FoxPro, Access îşi arătau limitele în materie de volum de date ce poate fi gestionat, securitate, păstrarea coerenţei în caz de avarie etc. Prin lansarea unor produse sub eticheta *Workgroup Databases,* marii producători de SGBDR-uri au creat o anumită confuzie în rândul utilizatorilor şi chiar al distribuitorilor.

In ultimul timp, SGBDR-urile pentru grupuri de lucru au fost asimilate celor de „categoria grea : Oracle, DB2, Sybase, Informix, SQL Server etc. Acestea sunt orientate pe gestiunea informatizată a organizaţiilor mari şi foarte mari, asigurându-se accesul simultan a sute, chiar mii de utilizatori la o aceeaşi bază de date. Dacă, tradiţional, piaţa acestui segment era rezervată Unix-ului, în prezent Windows NT şi Novell NetWare fac faţă celor
mai multe cerinţe. ...

Se mai cuvine adăugat că eforturi substanţiale sunt canalizate pe linia unui mariaj al relaţionalului de obiectual. Includerea de noi tipuri de date, extensii ale nucleului relaţional „tradiţional” al bazelor de date către obiecte, suportul pentru Web şi comerţ electronic sunt doar câteva direcţii pe care s-au înscris majoritatea SGBDR-urilor actuale pentru a-şi conserva sau ameliora poziţia de pe piaţă.

---

7 Vezi T. L6vy-Abegnoli, „SGBD Workgroup: un concept tres marketing”, *01 Informatique,* nr. 1354/1995, pp. 24-25; „Workgroup Databases Redux” (M, Ricciuti), *InfoWorld,* voi. n’ nr. 46/1995, p. 43.

1.3. Relaţii/tabele, domenii şi atribute

La modul simplist, o bază de date relaţională (BDR) poate fi definită ca un ansamblu de relaţii (tabele); fiecare tabelă (sau tabel), alcătuită din linii (tupluri), are un nume unic şi este stocată pe suport extern (de obicei disc). La intersecţia unei linii cu o coloană se găseşte o valoare atomică.

O relaţie conţine informaţii omogene legate de anumite entităţi, procese, fenomene: CĂRŢI, STUDENŢI, LOCALITĂŢI, PERSONAL, FACTURI etc. Spre exemplu, în figura 1.6 este reprezentată tabela JUDEŢE.

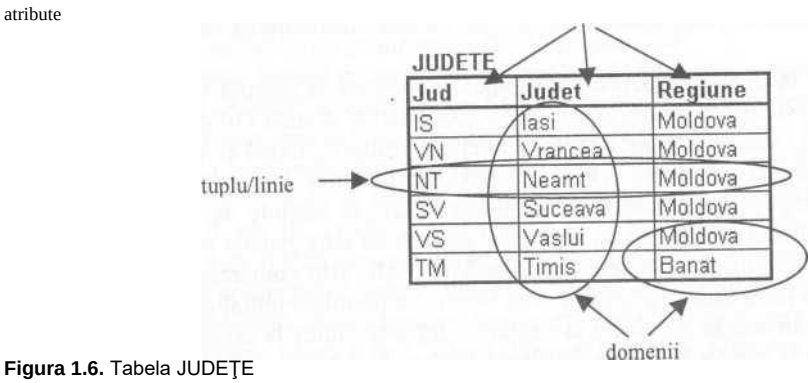


Figura 1.6. Tabela JUDEŢE

în teoria relaţională se foloseşte termenul *relaţie.* Practica, însă, a consacrat termenul *tabelă* (engl. *table).* Pentru inducerea în cititor a senzaţiei de pragmatism, cu care a fost impregnată prezenta lucrare, în SQL vom folosi preponderent noţiunea de tabelă. Cei care au de gând să scrie o lucrare teoretică (inevitabil, ştiinţifică!) sunt consiliaţi să apeleze la celălalt termen.

Un *tuplu* sau o *linie* este o succesiune de valori de diferite tipuri. In general, o linie regrupează informaţii referitoare la un obiect, eveniment etc., altfel spus, *informaţii referitoare la o entitate',* o carte (un titlu sau un exemplar din depozitul unei biblioteci, în funcţie de circumstanţe), un/o student(ă), o localitate (oraş sau comună), un angajat al firmei, o factură emisă etc. Figura 1.7 conţine al treilea tuplu din tabela JUDEŢE, tuplu referitor la judeţul Neamţ.

NT _____ Neamţ _____ Moldova ~|

Figura 1.7. Un tuplu al tabelei JUDEŢE

Linia de mai sus este alcătuită din trei valori ce desemnează indicativul (auto) al judeţului, numele judeţului şi regiunea din care face parte (una dintre provinciile istorice), valori specifice judeţului Neamţ.

Teoretic, orice tuplu reprezintă o *relaţie între clase de valori* (în cazul nostru, între trei clase de valori); de aici provine sintagma *baze de date relaţionale,* în sensul matematic al relaţiei, de asociere a două sau mai multe elemente.

Fireşte, toate tuplurile relaţiei au acelaşi format (structură), ceea ce înseamnă că în tabela JUDEŢE, din care a fost extrasă linia din figura 1.7, fiecare linie este constituită dintr-o valoare a indicativului, o valoare a denumirii şi o valoare a regiunii.

Ordinea tuplurilor nu prezintă importanţă din punctul de vedere al conţinutului informaţional al tabelei.

Ansamblul valorilor de acelaşi tip corespunde unui *atribut* al entităţilor, atribut ale cărui valori

alcătuiesc una dintre coloanele tabelei. Tabela din figura 1.6 prezintă trei coloane; prin urmare, se spune că relaţia JUDEŢE conţine trei atribute: Jud, Judeţ şi Regiune.

Fiecare atribut este caracterizat printr-un *nume* şi un *domeniu* de valori pe care le poate lua. *Domeniul* poate fi definit ca *ansamblul valorilor acceptate (autorizate) pentru un element component al relaţiei:*

- într-o tabelă destinată datelor generale ale angajaţilor, pentru atributul Sex, domeniul este alcătuit din două valori: *Femeiesc* şi *Bărbătesc,*
- pentru atributul Regiune, domeniul, deşi limitat, este ceva mai mare; valorile autorizate pot fi: *Dobrogea, Banat, Transilvania, Oltenia, Muntenia, Moldova.*
- domeniul atributului Jud este alcătuit din indicativele auto (două caractere) ale fiecărui judeţ (plus Bucureşti).
- domeniul unui atribut precum PreţUnitar, care se referă la preţul la care a fost vândut un produs/serviciu, este cu mult mai larg, fiind alcătuit din orice valoare cuprinsă între 1 şi 999999999 lei (depinde de inflaţie şi, implicit, de FMI).

Fireşte, în funcţie de specificul bazei de date, domeniul poate fi extins sau restrâns, după cerinţe. De exemplu, la regiunile luate în discuţie mai sus mai pot fi adăugate şi provincii precum: Bucovina, Maramureş, Crişana.

Integritatea domeniului priveşte controlul sintactic şi semantic al unei date oarecare şi face referinţă la modul său de definire.

Două domenii sunt declarate *compatibile* dacă sunt comparabile din punct de vedere semantic şi sintactic, adică ansamblurile care le definesc nu sunt disjuncte. Prin definiţie, două domenii identice sau incluse unul în celălalt sunt compatibile.

Pentru două tabele: FURNIZORI cu atributele: Nume, Adresa şi Localitate şi CLIENŢI cu atributele: Nume, Adresa, Localitate şi CodFiscal, se poate spune că domeniile atributelor Nume sunt compatibile (ambele cuprind mulţimea firmelor româneşti; de fapt, un şir de caractere de o anumită lungime). Acelaşi lucru se poate spune despre domeniile atributelor Adresa şi Localitate ale celor două relaţii.

După cum vom vedea într-un alt capitol, algebra relaţionala (şi SQL-ul) utilizează conceptul de domeniu compatibil pentru operatorii reuniune, intersecţie şi diferenţă.

Dacă notăm cu D, domeniul atributului Jud, cu D_2 domeniul atributului Judeţ şi cu D_3 domeniul pentru Regiune, se poate spune că fiecare linie a tabelei JUDEŢE este un tuplu de patru elemente:

$$(v_p\ Vj, v_3), \text{ unde v, eD,,} v_2 e D_2 \text{ şi v, e } D_3$$

Relaţia JUDEŢE corespunde unui subansamblu din ansamblul tuturor tuplurilor posibile alcătuite din patru elemente, ansamblu care este produsul cartezian

$$\underset{i=l}{x}D,$$

în general, orice relaţie R poate fi definită ca un subansamblu al produsului cartezian de *n* domenii D,:

$$RcD, xD, xD, x...xD_n ,$$

n fiind denumit *gradul* sau *ordinul* relaţiei. Relaţiile de grad 1 sunt *unare,* cele de grad 2 - *binare, ...,* cele de grad *n* - *n-are.* Această definiţie pune în evidenţă aspectul constant al relaţiei, de independenţă în timp (se spune că în acest caz relaţia este definită ca un *predicat).*

O a doua definiţie abordează o relaţie R ca un *ansamblu* de w-uplete (/w-tupluri) de valori:

$$R = \{t,,t_2,...,t_k,...,t_m\}, \text{ unde } t_k = (d_{kl},d_{k2},...,d_{ki},...,dj,$$

în care:

dk, este o valoare în D_{lt} dk_2 este o valoare în D_2, ... , dk_neste o valoare în D_n;

n - reprezintă ordinul lui R;

m - cardinalitatea lui R.

Pentru un plus de claritate, vezi figura 1.8.

Reprezentarea sub formă de tabelă, deci ca ansamblu de tupluri, pune în evidenţă aspectul dinamic, variabil al relaţiei. Abordarea predicativă (prima definiţie) sau asamblistă (a doua definiţie) reprezintă un criteriu important de delimitare a limbajelor de interogare a bazelor de date relaţionale.

Având în vedere cele două definiţii ale relaţiei, se poate reformula conceptul de *atribut,* ca fiind elementul *care determină ansambluI de valori luate de fiecare domeniu al relaţiei*[5]. Precizarea fără ambiguitate a rolului jucat de fiecare domeniu în cadrul relaţiei impune ca numele atributelor să fie diferite. O relaţie poate fi simbolizată şi astfel:

$$R(A_{,}, A_2,...,A_{,,}) \text{ sau } (A_{,}: D_{,}, A_2 : D_2,...,A_{,,}: D_n) \text{ , unde}$$

Ai sunt atributele relaţiei.

R

	^{p}i	**Da**	**D_3**		**D,**		**D_n**
t,	dl1	di2	^{d}13		dii		^{d}1n
tî	d_2i	$d_2$2	^{d}23		^{d}2i		^{d}2n
t	^{d}31	^{d}32	^{d}33		^{d}3i		^{d}3n
3							
t k	^{d}k1	^{d}k2	^{d}k3		^{d}ki		^{d}kn
tr a	^{d}m1	^{d}m2	^{d}m3		^{d}mi		^{d}mn

Figura 1.8. Ilustrarea celei de-a doua definiţii a unei relaţii

Reţinem corespondenţa noţiunilor *relaţie-tabelă, tuplu-linie* şi *atribut-coloană.*

Numărul de tabele pe care le conţine o bază de date, atributele „adunate" în fiecare tabelă, domeniul fiecăruia dintre atribute prezintă diferenţe majore de la o bază la alta, chiar dacă uneori reflectă acelaşi tip de procese. Intrăm astfel în sfera proiectării bazelor de date, a dependenţelor şi normalizării.

înainte de finalul paragrafului, să examinăm o altă tabelă, CLIENŢI, prezentată în figura 1.9.

CLIENŢI

Cod Cl	DenCl	CodFiscal	Adresa	CodPost	Telefon
1001	Client 1 SRL	R1001	Tranziţiei, 13 bis	6600	NULL
1002	Client 2 SA	R1002	NULL	6600	032-212121
1003	Client 3 SRL	R1003	Prosperităţii, 22	6500	035-222222
1004	Client 4	NULL	Sapientei, 56	5725	NULL
1005	Client 5 SRL	R1005	NULL	1900	056-111111
1006	Client 6 SA	R1006	Pacientei, 33	5550	NULL
1007	Client 7 SRL	R1007	Victoria Capitalismului, 2	1900	056-121212

Figura 1.9. Valori NULLe într-o relaţie

Relaţia conţine informaţii despre clienţii firmei (firma este cea pentru care s-a proiectat, creat şi implementat baza de date). Fiecare linie se referă la un singur client (persoană juridică). Se observă că atributele CodFiscal, Adresa şi Telefon conţin, pe alocuri, o valoare curioasă notată NULL. Valoarea NULL este considerată o metavaloare şi indică faptul că, în acel loc, informaţia este necunoscută sau inaplicabilă. în tabela CLIENŢI, Clientului 1 SRL (cod 1001) nu i se cunoaşte telefonul, pentru Client 2 SA nu se cunoaşte adresa, pentru Client 4 (cod 1004) nu se cunosc nici codul fiscal, nici telefonul etc.

Pentru cei deprinşi să lucreze cu SGBD-uri sub MS-DOS, valorile respective sunt fie 0 (pentru atribute numerice), fie ‘ ‘ (pentru şiruri de caractere), fiind extrase din tabele prin funcţii precum EMPTY ().

Valoarea NULL este diferită însă (cel puţin teoretic) de valorile 0 sau spaţiu. Importanţa este majoră în expresii şi funcţii, după cum vom vedea în capitolele consacrate limbajului SQL.

în încheiere, principalele caracteristici ale unei relaţii sunt sistematizate după cum urmează:

- « în cadrul unei baze de date, o relaţie prezintă un nume distinct de al celorlalte relaţii.
- Valoarea unui atribut într-un tuplu oarecare conţine o singură valoare (o valoare atomică).
- Fiecare atribut are un nume distinct.
- Orice valoare a unui atribut face parte din domeniul pe care a fost definit acesta.
- Ordinea dispunerii atributelor în relaţie nu prezintă importanţă.
- Fiecare tuplu este distinct, adică nu pot exista două tupluri identice.
- Ordinea tuplurilor nu influenţează conţinutul informaţional al relaţiei.

Facem astfel trecerea la următorul paragraf, dedicat restricţiilor unei baze de date relaţionale.

1.4. Restricţii ale bazei de date

Există diverse moduri de abordare şi clasificare a restricţiilor ce pot fi instituite într-o bază de date. în cele ce urmează vor fi prezentate: restricţia de domeniu, de atomicitate, de unicitate, referenţială şi restricţiile-utilizator.

Restricţia de domeniu

După cum am văzut în paragraful anterior, un atribut este definit printr-un nume şi un domeniu. Orice valoare a atributului trebuie să se încadreze în domeniul definit. Există mai multe moduri de percepţie a acestei restricţii.

O parte dintre informaticieni substituie domeniul tipului atributului: numeric, şir de caractere, dată calendaristică, logic (boolean) etc. şi, eventual, lungimii (numărul maxim de poziţii/caractere pe care se poate „întinde" un atribut). După cum se observă, este luat în calcul numai aspectul sintactic al domeniului. Faptul că indicativul unui judeţ poate fi una dintre valorile: IS, TM, B etc. reprezintă o *restricţie de comportament* sau, mai simplu, o *restricţie definită de utilizator.*

Cea de-a doua categorie priveşte domeniu! deopotrivă sintactic şi semantic. Astfel, domeniul sintactic al atributului Jud (indicativul judeţului) este un şir de două caractere, obligatoriu litere (sau o literă şi un spaţiu, pentru Bucureşti), şi, chiar mai restrictiv, literele sunt obligatoriu majuscule. Din punct de vedere semantic, indicativul poate lua una dintre valorile: IS, TM...

Majoritatea SGBD-urilor permit definirea tuturor elementelor ce caracterizează domeniul (sintactic şi semantic) atributului Jud prin declararea tipului şi lungimii atributului şi prin aşa-numitele reguli de validare la nivel de câmp *(field validation rule).* Sunt însă şi produse la care domeniul poate fi definit explicit, sintactic şi semantic, dându-i-se un nume la care vor fi legate atributele în momentul creării tabelelor.

Atomicitate

Conform teoriei bazelor de date relaţionale, orice atribut al unei tabele oarecare trebuie să fie *atomic,* în sensul imposibilităţii descompunerii sale în alte atribute. Implicit, toate domeniile unei baze de date sunt musai atomice (adică elementare). în aceste condiţii, se spune că baza de date se află în *prima formă normală* sau *prima formă normalizată* (INF).

Astăzi, atomicitatea valorii atributelor a devenit o ţintă predilectă a „atacurilor duşmănoase la adresa modelului relaţional, datorită imposibilităţii înglobării unor structuri de date mai complexe, specifice unor domenii ca: proiectare asistată de calculator, baze de date multimedia etc. Mulţi autori, dintre care merită amintiţi cu deosebire Chris J. Date şi Hugh Darwen, se opun ideii de atomicitate formulată de Codd.

Primele vizate de „stigmatul" neatomicităţii sunt atributele compuse. Un exemplu de atribut compus (non-atomic) este Adresa. Fiind alcătuită din Stradă, Număr, Bloc, Scară, Etaj, Apartament, discuţia despre atomicitatea adresei pare de prisos, iar descompunerea sa imperativă. Trebuie să ne raportăm însă la obiectivele bazei. Fără îndoială că pentru BD a unei filiale CONEL sau ROMTELECOM, sau pentru poliţie, preluarea separată a fiecărui element constituent al adresei este foarte importantă. Pentru un importator direct însă, pentru un mare en-grosist sau pentru o firmă de producţie lucrurile stau într-o cu totul altă lumină. Partenerii de afaceri ai acestora sunt persoane juridice, iar adresa interesează numai la nivel general, caz în care atributul Adresa nu este considerat a fi non-atomic.

Alte exemple de atribute ce pot fi considerate, în funcţie de circumstanţe, simple sau compuse: DataOperaţiuniiBancare (Data+Ora), Buletinldentitate (Seria+Număr), NrînmatriculareAuto (privit global sau pe cele trei componente: număr, judeţ, combinaţie trei de litere).

O relaţie (tabelă) în INF nu trebuie să conţină atribute care se repetă ca grupuri *(grupuri repetitive).* Intr-o altă formulare, toate liniile unei tabele trebuie să conţină acelaşi număr de atribute. Fiecare celulă a tabelei (intersecţia unei coloane cu o linie) - altfel spus, valoarea unui atribut pe o linie (înregistrare) - trebuie să fie atomică.

După [Connoly ş.a. 96], un *grup repetitiv* este un atribut sau grup de atribute dintr-o tabelă care apare cu valori multiple pentru o singură apariţie a cheii primare a tabelei nenormalizate.

Să luăm exemplul tabelei BIBLIOTECĂ din figura 1.10. Tabela gestionează informaţii despre cărţile existente în depozitul bibliotecii facultăţii X. Depozitul conţine două exemplare ale cărţii cu ISBN 973-333-566. Cartea a fost scrisă de trei autori şi îi sunt asociate patru cuvinte-cheie. Relaţia nenormalizată conţine trei grupuri repetitive: Cote, Autori şi CuvinteCheie.

BIBLIOTECA

ISBN	Titlu	Cote	Autori	Editura	An	CuvinteCheie
973-333-566	Instrumente CASE	III 13421 III 13422	Oprea Dumitriu Meşniţă	Alfa	1998	Analiză SI Proiectare SI CASE DFD

Figura 1.10. Relaţie în care există valori ne-atomice

Restricţia de unicitate

într-o relaţie nu pot exista două linii identice (două linii care prezintă aceleaşi valori pentru toate atributele). Mai mult, majoritatea relaţiilor prezintă un atribut, sau o combinaţie de atribute, care diferenţiază cu siguranţă un tuplu de toate celelalte tupluri ale relaţiei.

Cheia primară a unei relaţii (tabele) este un atribut sau un grup de atribute care identifică ftră ambiguitate fiecare tuplu (linie) al relaţiei (tabelei).

După Date, există trei restricţii pe care trebuie să le verifice cheia primară:

1. *Unicitate*: o cheie identifică un singur tuplu (linie) al relaţiei.
2. *Compoziţie minimală*: atunci când cheia primară este compusă, nici un atribut din cheie nu poate fi eliminat fară distrugerea unicităţii tuplului în cadrul relaţiei. în cazuri-limită, o cheie poate fi alcătuită din toate atributele relaţiei.
3. *Valori non-nule*: valorile atributului (sau ale ansamblului de atribute) ce desemnează cheia primară sunt întotdeauna specificate, deci nenule. Mai mult, nici un atribut din compoziţia cheii primare nu poate avea valori nule. Această a treia condiţie se mai numeşte şi *restricţie a entităţii.*

Domeniul unui atribut care este cheie primară într-o relaţie este denumit *domeniu primar.* Dacă într-o relaţie există mai multe combinaţii de atribute care conferă unicitate tuplului, acestea sunt denumite *chei candidate.* O cheie candidată care nu este identificator primar este cunoscută şi sub numele de *cheie alternativă.*

Celko inventariază patru proprietăţi dezirabile pentru o cheie[17]:

- Familiaritate: valorile cheii să fie uşor de înţeles pentru utilizatori.
- Stabilitate: valorile cheii nu trebuie să fie volatile.
- Minimalitate.
- Simplitate: sunt de preferat chei scurte şi simple.

Tot el atrage atenţia asupra faptului că cele patru proprietăţi pot intra uneori în conflict. O cheie stabilă poate să se dovedească complexă şi dificil de gestionat. Identificarea cheii primare pentru o relaţie face parte (împreună cu stabilirea tabelelor prin gruparea atributelor, stabilirea celorlalte restricţii) din procesul de proiectare a bazei de date. Uneori, precum în cazul tabelei JUDEŢE, stabilirea cheii primare nu ridică probleme. Cum fiecare judeţ are un indicativ auto unic, rezultă că atributul Jud candidează la postul de cheie a relaţiei. La fel se poate spune şi despre denumirea judeţului. Rezultă că relaţia JUDEŢE prezintă două chei candidate: Jud şi Judeţ. Alegerea uneia dintre cheile candidate are în vedere criterii precum lungimea, uşurinţa în reţinere şi/sau editare. Fiind mai scurt, desemnăm atributul Jud drept cheie primară, situaţie în care Judeţ devine cheie alternativă..

Există suficiente cazuri în care cheia primară este compusă din două, trei ş.a.m.d. sau, la extrem,

17. [Celko99], p. 247.

toate atributele relaţiei. Să luăm spre analiză o relaţie PERSONAL care conţine date generale despre angajaţii firmei. Fiecare tuplu al relaţiei se referă la un angajat, atributele fiind: Nume, Prenume, DataNaşterii, Vechime, SalariuTarifar.

- Atributul Nume nu poate fi cheie, deoarece chiar şi într-o întreprindere de talie mijlocie, este posibil să existe doi angajaţi cu acelaşi nume.
- Dacă apariţia a două persoane cu nume identice este posibilă, atunci apariţia a două persoane cu acelaşi Prenume este probabilă.
- Nici unul dintre atributele DataNaşterii, Vechime, SalariuTarifar nu poate fi „înzestrat"'cu funcţia de identificator.
- In acest caz, se încearcă gruparea a două, trei, patru ş.a.m.d. atribute, până când se obţine combinaţia care va permite diferenţierea clară a oricărei linii de toate celelalte.
- Combinaţia Nume+Adresă pare, la prima vedere, a îndeplini „cerinţele" de identificator. Ar fi totuşi o problemă: dacă în aceeaşi firmă (organizaţie) lucrează împreună soţul şi soţia? Ambii au, de obicei, acelaşi nume de familie şi, tot de obicei, acelaşi domiciliu. Este adevărat, cazul ales nu este prea fericit. Dar este suficient pentru „compromiterea" combinaţiei.
- Următoarea tentativă este grupul Nume + Prenume+Adresă, combinaţie neoperantă dacă în organizaţie lucrează tatăl şi un fiu (sau mama şi o fiică) care au aceleaşi nume şi prenume şi domiciliul comun.
- Ar rămâne de ales una dintre soluţiile: Nume + Prenume+Adresă+Vechime, Nume+Prenume+Adresă+DataNaşterii.

Oricare dintre cele două combinaţii prezintă riscul violării restricţiei de entitate, deoarece este posibil ca, la preluarea unui angajat în bază, să nu i se cunoască adresa sau data naşterii, caz în care atributul respectiv ar avea valoarea NULL.

Dificultăţile de identificare fără ambiguitate a angajaţilor au determinat firmele ca, la angajare, să aloce fiecărei persoane un număr unic, denumit Marcă. Prin adăugarea acestui atribut la cele existente, pentru relaţia PERSONAL problema cheii primare este rezolvată mult mai simplu. Actualmente, sarcina este simplificată şi prin utilizarea codului numeric personal (CNP), combinaţie de 13 cifre care prezintă avantajul că rămâne neschimbată pe tot parcursul vieţii persoanei (şi după, dar asta are mai puţină importanţă).

Problema unicităţii este una discutabilă. Clasicii modelului relaţional, Codd şi Date, s-au pronunţat împotriva repetării tuplurilor în cadrul unei relaţii, cerinţă pe care standardele SQL nu au luat-o în considerare. Există şi situaţii reale în care repetarea liniilor este inevitabilă. David Beech a prezentat într-o lucrare înaintată Consiliului de Standardizare a bazelor de date ANSI X3H2, şi mai apoi în revista *Datamation*[IS], problema consacrată în literatura de specialitate drept „argumentul cutiilor cu mâncare pentru pisici"[8]: într-un supermagazin în care casele de marcat sunt legate la o bază de date, un client cumpără patru cutii Whiskas pentru pisici care costă, fiecare (cutie), 10.000 lei; pe bon apare de patru ori
„cutii Whiskas 10000". Autorul afirmă, pe bună dreptate, că la nivelul conceptual la care se înregistrează informaţiile, nu există nici o valoare prin care cele patru rânduri să fie diferenţiate. Astfel încât, pentru a respecta canoanele modelului relaţional, este necesară introducerea unei coloane suplimentare fără prea mare relevanţă, dar care să asigure unicitatea fiecărui tuplu.

Este adevărat că E.F. Codd a propus ulterior consiliului ANSI X3H2 o funcţie specială care să măsoare gradul de repetabilitate a valorii unuia sau mai multor atribute, funcţie care ar semăna întru câtva cu funcţia COUNT () din SQL folosită împreună cu GROUP BY'°. Dacă am privi cazul celor patru conserve Whiskas prin prisma celor trei abordări, SQL le priveşte ca patru cutii separate, dar echivalente, modelul Codd (original)/Date le consideră o singură clasă de cutii, iar pentru ultimul model al lui Codd avem de-a face cu o colecţie de patru cutii.

8 [Celko99], p. 45.

Pe lângă noţiunile cheie candidată, cheie primară, cheie alternativă, modelul relaţional utilizează şi sintagma *supercheie.* După Celko"[1], o supercheie poate fi definită ca un set de coloane ce îndeplineşte condiţia de cheie (unicitate), dar care conţine cel puţin un subset care este el însuşi cheie. Cu alte cuvinte, din cele trei condiţii ale cheii primare, o supercheie o îndeplineşte numai pe prima (unicitate), fără a avea însă compoziţia minimală şi fără a pune problema restricţiei de entitate (valorile nule ale atributelor componente).

Restricţia referenţială

O bază de date relaţională este alcătuită din relaţii (tabele) aflate în legătură. Stabilirea legăturii se bazează pe mecanismul cheii străine şi, implicit, al restricţiei referenţiale. Figura 1.11 prezintă o bază de date formată din două tabele, LOCALITĂŢI şi JUDEŢE.

Atributul Jud joacă un rol de „agent de legătură" între tabelele LOCALITĂŢI şi JUDEŢE. Pentru relaţia JUDEŢE, atributul Jud este cheie primară, în timp ce în tabela LOCALITĂŢI, Jud reprezintă *coloana de referinţă* sau *cheia străină,* deoarece numai pe baza valorilor sale se poate face legătura cu relaţia JUDEŢE.

Cheile străine sau coloanele de referinţă sunt deci atribute sau combinaţii de atribute care pun în legătură linii (tupluri) din relaţii diferite. Tabela în care atributul de legătură este primară se numeşte *tabelă-părinte* (în cazul nostru JUDEŢE), iar cealaltă *tabelă-copil.*

LOCALITATI

CodPost	Loc	Jud
6600	Iasi	IS <—
5725	Paşcani	IS <—
6500	Vaslui	vs ^
5300	Focşani	VN
6400	Birlad	VS
5800	Suceava	SV
5550	Roman	NULL
1900	Timişoara	TM /\

JUDEŢE

Jud	Judeţ	Regiune
IS	Iasi	Moldova
— VN	Vrancea	Moldova
NT	Neamţ	Moldova
-- SV	Suceava	Moldova
^VS	Vaslui	Moldova
—TM	Timiş	Banat

Figura 1.11. Mecanismul de legare a tabelelor

Legat de noţiunea de cheie străină apare conceptul de *restricţie referenţială.* O restricţie de integritate referenţială apare atunci când o relaţie face referinţă la o altă relaţie. C.J. Date defineşte restricţia de integritate referenţială astfel[22]:

Fie D un domeniu primar şi R1 o relaţie ce prezintă un atribut A definit pe D - se poate scrie Rl(..., A:D, ...). *In orice moment, orice valoare a lui A in R1 trebuie să fie:*

a) ori *nulă,*

b) ori *egală cu o valoare a lui V, unde V este cheie primară sau candidată într-un tuplu al unei relaţii oarecare R2 (R1 şi R2 nu trebuie să fie neapărat distincte), cheie primară definită pe acelaşi domeniu primar D* - R2(V:D, ...).

într-o altă formulare, se iau în discuţie două tabele (relaţii) TI şi T2. Ambele prezintă atributul sau grupul de atribute notat *CH,* care *pentru TI este cheie primară, iar pentru T2 cheie străină. Dacă în T2 se interzice apariţia de valori nenule ale CH care nu există în nici un tuplu din TI, se spune că între T2 şi TI s-a instituit o restricţie referenţială.*

Instituirea restricţiei referenţiale între tabela JUDEŢE (părinte) şi LOCALITĂŢI (copil) permite cunoaşterea, pentru fiecare localitate, a denumirii judeţului şi a regiunii din care face parte. Dacă în LOCALITĂŢI ar exista vreo linie în care valoarea atributului Jud ar fi, spre exemplu, XY, este clar că acea linie ar fi orfană (nu ar avea linie corespondentă în tabela-părinte JUDEŢE).

Observaţii

1. *Pentru mulţi utilizatori şi profesionişti ai bazelor de date, denumirea de „relaţional" desemnează faptul că o bază de date este alcătuită din tabele puse în legătură prin intermediul cheilor străine. Aceasta este, de fapt, a doua accepţiune a termenului de BDR, prima, cea „clasică", având în vedere percepţia fiecărei Unii dintr-o tabelă ca o relaţie între clase de valori.*
2. *Majoritatea SGBD-urilor prezintă mecanisme de declarare şi gestionare automată a restricţiilor referenţiale, prin actualizări în cascadă şi interzicerea valorilor care ar încălca aceste restricţii.*
3. *Respectarea restricţiilor referenţiale este una dintre cele mai complicate sarcini pentru dezvoltatorii de aplicaţii ce utilizează baze de date. Din acest punct de vedere, tentaţia este de a „sparge" baza de date în cât mai puţine tabele cu putinţă, altfel spus, de a avea relaţii cât mai „corpolente". Gradul de fragmentare al bazei ţine de* normalizarea *bazei de date, care, ca parte a procesului de proiectare a BD, se bazează pe dependenţele funcţionale, multivaloare şi de joncţiune între atribute.*

Restricţii-utilizator

Restricţiile-utilizator mai sunt denumite şi restricţii de comportament sau restricţii ale organizaţiei. De obicei aceste restricţii iau forma unor reguli de validare la nivel de atribut, la nivel de linie/tabelă sau al unor reguli implementate prin declanşatoare (*triggere*). Spre exemplu, se poate institui o regulă care interzice emiterea unei noi facturi (o nouă vânzare) dacă datoriile firmei-client sunt mai mari de 2.000.000.000 lei, iar directorul acesteia nu este membru în partidul de guvernământ sau coaliţia de guvernare.

1.5. Schema şi conţinutul unei baze de date. Exemplu

Există două aspecte complementare de abordare a bazelor de date relaţionale: *schema* (structura, intensia) şi *conţinutul* (instanţierea, extensia)[9].

Conţinutul unei relaţii este reprezentat de ansamblul tuplurilor ce o alcătuiesc la un moment dat. Pe parcursul exploatării bazei, conţinutul poate creşte exponenţial, în funcţie de volumul şi complexitatea operaţiunilor consemnate.

O schemă relaţională poate fi definită ca un ansamblu de relaţii asociate *semantic* prin domeniul lor de definiţie şi prin restricţii de integritate[10]. Este independentă de timp şi reprezintă componenta permanentă a relaţiilor.

După C.J. Date, schema bazei de date (intensia) cuprinde[2]':

- *Structura titulaturilor* alcătuită din numele relaţiilor şi cele ale atributelor (fiecare atribut fiind asociat domeniului său) şi
- *Restricţiile de integritate,* care sunt de trei feluri:
  a) *Restricţiile cheilor primare.* După cum a fost prezentat, fiecare cheie primară este supusă restricţiilor de unicitate, compoziţie minimală şi valori nenule.
  b) *Restricţii referenţiale,* care decurg din existenţa cheilor străine.
  c) *Alte restricţii.* în această categorie sunt incluse restricţiile definite de utilizator (de comportament), dependenţele dintre atribute etc.

De multe ori este suficientă cunoaşterea numai a schemei simplificate. *Schema simplificată* a unei baze de date relaţionale cuprinde numele tabelelor şi enumerarea atributelor acestora, atributele-

9 în multe dintre lucrările redactate în limba engleză se folosesc noţiunile de „intension" şi „extension" (spre exemplu, vezi [Date96]). De aici traducerea „intensie" şi „extensie" din [Lungu ş.a. 95].

10 [Saleh94], p. 29.**Figura** 1.12. Schema simplificată a bazei de date VÂNZĂRI

chei primare fiind sublimate.

în capitolele care vor urma, va fi utilizată cu precădere o bază de date „martor" sau, mai bine spus, „cobai", denumită VÂNZĂRI, ce conţine următoarele tabele:

JCJDETE {Jud, Judeţ, Regiune}
LOCALITATI {CodPost, Loc, Jud)
PERSOANE{CNP, Nume, Prenume, Adresa, Sex, CodPost, TelAcasa, TelBirou, TelMobil, EMail}
CLIENŢI{CodCl, DenCl, CodFiscal, Adresa, CodPost, Telefon} PERSCLIENTI{CNP, CodCl, Funcţie}
PRODUSE {CodPr, DenPr, UM, Grupa,. ProcTVA}
FACTURI{NrFact, DataFact, CodCl, Obs}
LINIIFACT{NrFact, Linie, CodPr, Cantitate, PretUnit}
INCASARI{Codlnc, Datalnc, CodDoc, NrDoc, DataDoc}
INCASFACT{Codlnc, NrFact, Transa}

Discuţia detaliată despre fiecare o amânăm peste numai câteva zeci de rânduri, când, pentru un plus de claritate, va fi prezentat şi conţinutul fiecăreia.

Schema unei baze de date poate fi reprezentată grafic, avantajul principal fiind o mai bună sugestivitate. Iată câteva reguli pentru reprezentarea grafică a unei BDR[26]:

- O tabelă se reprezintă pe două linii, pe prima fiind scris numai numele relaţiei, iar pe cea de-a doua numele atributelor.
- Ordinea prezentării atributelor nu are importanţă. Totuşi, în scopul uşurării înţelegerii schemei, coloanele care desemnează cheia primară se dispun succesiv (bineînţeles, atunci când sunt mai multe). La fel şi coloanele care constituie o cheie străină. în general, cheia primară este plasată la marginea stângă a tabelei (prima sau primele coloane).
- Numele coloanelor ce alcătuiesc cheia primară se subliniază cu o linie continuă, iar cele care alcătuiesc identificatorii secundari se subliniază cu linie punctată.
- Numele unei coloane facultative este scris între paranteze.
- Dacă o cheie străină este alcătuită din mai multe coloane, se utilizează acolada pentru a le grupa.
- O restricţie referenţială este reprezentată printr-o săgeată care pleacă de la numele coloanei de referinţă (sau de la vârful acoladei, în cazul unui grup de referinţă) şi are vârful în atributele tabelei la care se face referinţa (tabela-părinte).

Pentru baza de date VÂNZĂRI, reprezentarea grafică a schemei simplificate este cea din figura 1.12.

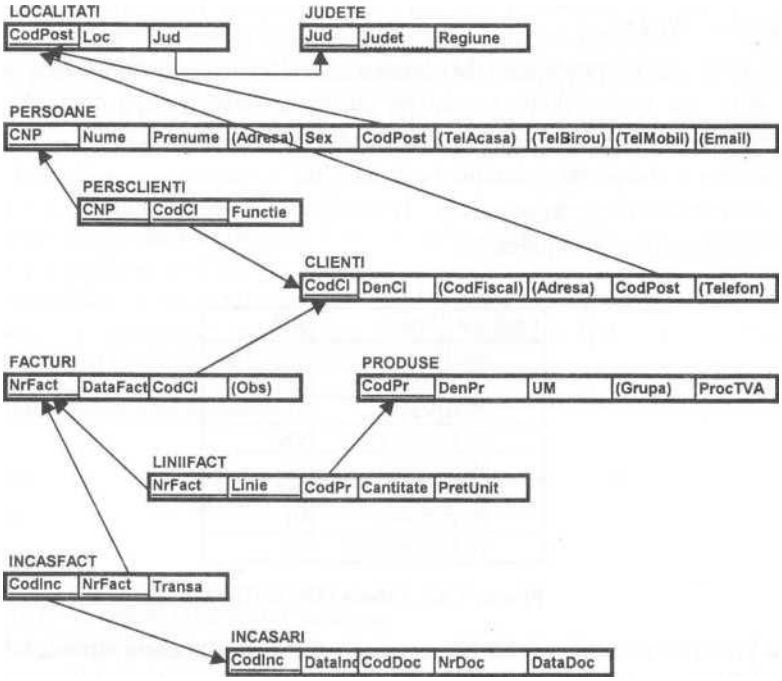


Figura 1.12. Schema simplificată a bazei de date VÂNZĂRI

Tabela **JUDEŢE** (figura **1.13)** conţine informaţii generale despre judeţele în care sunt clienţi. Fiecare linie a tabelei descrie un judeţ. Atributele sunt:

- Jud - indicativul auto al judeţului (alcătuit din două litere, majuscule);
- Judeţ - denumirea judeţului;
- Regiune - regiunea istorică (provincia) din care face parte judeţul.

JUDEŢE

Jud	Judeţ	Regiune
IS	Iasi	Moldova
VN	Vrancea	Moldova
NT	Neamţ	Moldova
SV	Suceava	Moldova
VS	Vaslui	Moldova
TM	Timiş	Banat

Figura 1.13. Tabela JUDEŢE

Cheia primară este atributul Jud. Atributul Judeţ este cheie alternativă. Pot fi instituite o serie de restricţii-utilizator:

- Jud este alcătuit numai din majuscule (eventual un spaţiu, pentru Bucureşti);
- Fiecare cuvânt din Judeţ începe cu majusculă; restul literelor sunt mici;
- Regiune începe cu majusculă; restulliterelor sunt mici;
- Regiune poate avea numai una din valorile: Banat, Dobrogea, Muntenia, Oltenia, Transilvania, Moldova.

Tabela **LOCALITĂŢI** (figura 1.14) conţine câte o linie pentru fiecare oraş sau comună. Atenţie: satele nu sunt preluate în această tabelă, deoarece, într-o comună, fiecare sat component are acelaşi cod poştal ca şi comuna. Atribute:

- CodPost - codul poştal al comunei sau oraşului;

26. [Hainaut94], pp. 24-25.

- Loc - denumirea comunei/oraşului;
- Jud - indicativul auto al judeţului.

LOCALITATI

CodPost	Loc	Jud
6600	Iasi	IS
5725	Paşcani	IS
6500	Vaslui	VS
5300	Focşani	VN
6400	Birlad	VS
5800	Suceava	SV
5550	Roman	NT
1900	Timişoara	TM

Figura 1.14. Tabela LOCALITAŢI

Cheia primară este atributul CodPost. Atributul Jud este cheie străină, tabela-părinte fiind JUDEŢE (prin atributul Jud). Pot fi instituite o serie de restricţii-utilizator:

- Jud este alcătuit numai din majuscule (eventual un spaţiu, pentru Bucureşti);

« Fiecare cuvânt din Localitate începe cu majusculă; restul literelor sunt mici.

Tabela **CLIENŢI** (figura 1.15) grupează datele generale ale clienţilor firmei pentru care s-a constituit baze de date (o linie - un client). Atribute:

- CodCl - codul clientului;
- DenCl - denumirea clientului (persoană juridică);
- CodFiscal - codul fiscal;
- Adresa - adresa sediului firmei-client;
- CodPost - codul poştal al comunei sau oraşului;
- Telefon - telefonul (principal) al clientului.

CLIENŢI

CodCl	DenCl	CodFiscal	Adresa	CodPost	Telefon
1001	Client 1 SRL	R1001	Tranziţiei, 13 bis	6600	NULL
1002	Client 2 SA	R1002	NULL	6600	032-212121
1003	Client 3 SRL	R1003	Prosperităţii, 22	6500	035-222222
1004	Client 4	NULL	Sapientei, 56	5725	NULL
1005	Client 5 SRL	R1005	NULL	1900	056-111111
1006	Client 6 SA	R1006	Pacientei, 33	5550	NULL
1007	Client 7 SRL	R1007	Victoria Capitalismului, 2	1900	056-121212

Figura 1.15. Tabela CLIENŢI

Cheia primară este atributul CodCl. Atributul CodPost este cheie străină, tabela- -părinte fiind LOCALITĂŢI (prin atributul CodPost). DenCl şi Adresa încep cu majuscule. După cum punctam anterior, valorile nule indică o lipsă de informaţie. Pentru primul client nu se cunoaşte telefonul, celui de-a! doilea adresa etc.

Tabela **PERSOANE** (figura 1.16) are ca obiectiv stocarea datelor despre persoanele- -cheie de la firmele-client: directori generali, directori financiari, şefi ai compartimentelor comerciale (aprovizionare şi/sau vânzări) etc. Probabil că vi se pare ciudat, dar ceea ce intenţionăm cu această tabelă (şi următoarea) este să sprijinim *fidelizarea* clientului. Nu costă (mai) nimic dacă de Sfântul Ioan se trimite câte o felicitare tuturor Ionilor şi Ioanelor din partea firmei noastre. Iar pentru ca lucrurile să fie şi mai bine puse la punct, ar fi trebuit să preluăm şi informaţii precum: data naşterii, starea civilă, numele şi vârsta copiilor, pasiuni în materie de muzică, arte pastice, literatură, sport etc. Schema luată în considerare s-a oprit la următoarele atribute:

- CNP - codul numeric al persoanei;
- Nume;
- Prenume;
- Adresa;
- Sex;
- CodPost;
- TelAcasă - numărul telefonului de acasă;
- TelBirou - numărul telefonului (fix) de la birou;
- TelMobil - numărul „mobilului";
- Email - adresa e-mail.

PERSOANE

CNP	Nume	Prenume	Adresa	Sex	CodPost	TelAcasa	TelBirou	TelMobil	EmaU
CNP1	Ioan	Vasile	! L.Caraqiale, 22	B	6600	123456	987654	094222222	NULL
CNP2	Vasile	Ion	NULL	B	6600	234567	876543	094222223	Ion@a.ro
CNP3	Popovici	Ioana	V.Micle, Bl.I. Sc.B,Ap.2	F	5725	345678	NULL	094222224	NULL
CNP4	Lazar	Caraion	M.Eminescu, 42	B	6500	456789	NULL	094222225	NULL
CNP5	Iurea	Simion	I.Creanqa, 44 bis	B	6500	567890	543210	NULL	NULL
CNP6	Vase	Simona	M.Eminescu, 13	F	5725	NULL	432109	094222227	NULL
CNP7	Popa	Ioanid	I.Ion. Bl.H2, Sc.C, Ap.45	B	1900	789012	321098	NULL	NULL
CNP8	Boqacs	Ildiko	I.V.Viteazu, 67	F	5550	890123	210987	094222229	NULL
CNP9	Ioan	Vasilica	Gării, Bl.B4, Sc.A, Ap.1	F	1900	901234	109876	094222230	NULL

Figura 1.16. Tabela PERSOANE

Deşi este un număr compus din 13 poziţii, este recomandabil ca CNP să fie declarat de tip şir de caractere. Aceasta deoarece în firmă există rezidenţi din Republica Moldova care au poziţii de vârf în firme româneşti (sau reprezentanţe ale unor firme din ţara natală). Or, codul identificator din paşaport este alcătuit dintr-o literă şi o serie de cifre.

Cheia primară este atributul CNP. Fiind vorba de persoane exterioare întreprinderii, un atribut precum Marca este mai puţin indicat. CNP este greoi, relativ lung (de aceea am preferat să scriem pur şi simplu CNP1, CNP2...), dar stabil şi unic.

Atributul CodPost este cheie străină, tabela-părinte fiind LOCALITĂŢI (prin atributul CodPost).

Celei de-a doua persoane nu i se cunoaşte domiciliul (adresa). Cu excepţia lui Vasile Ion, nimeni nu are cont deschis pentru poşta electronică.

Alte restricţii-utilizator:

- Fiecare cuvânt din Nume şi Prenume începe cu majusculă; restul literelor sunt mici;
- Adresa începe cu majusculă;
- Atributul Sex poate lua numai două valori: *B* de la *Bărbătesc* şi *F* de la *Femeiesc*.

Tabela PERSCLIENŢI (figura 1.17) indică funcţia deţinută de fiecare persoană la unui sau mai mulţi (deşi aceste cazuri sunt rare) clienţi. O linie din această tabelă reflectă o funcţie deţinută de o persoană la un client. O persoană poate avea mai multe funcţii, chiar la aceeaşi firmă (cumul de funcţii). Cele trei atribute sunt:

- CNP - persoana care deţine funcţia;
- CodCl - firma la care persoana deţine funcţia;
- Funcţie - funcţia deţinută.

PERSCLIENŢI

CNP	CodCl	Funcţie
CNP1	1001	Director general
CNP2	1002	Director general
CNP3	1002	Sef aprovizionare
CNP4	1003	Sef aprovizionare
CNP5	1003	Director financiar
CNP6	1004	Director qeneral
CNP7	1005	Sef aprovizionare
CNP8	1006	Director financiar
CNP9	1007	Sef aprovizionare

Figura 1.17. Tabela PERSCLIENŢI

Cheia primară este compusă din toate cele trei atribute: CNP+CodCl+Functie, aceasta deoarece am convenit că o aceeaşi persoană poate lucra la aceeaşi firmă pe mai multe funcţii. CNP şi CodCl sunt chei străine, tabelele-părinte fiind PERSOANE, respectiv CLIENŢI. Atributul Funcţie începe cu majusculă, restul literelor fiind mici.

Tabela PRODUSE (figura 1.18) reprezintă nomenclatorul de produse şi servicii pe care le comercializează firma (produsele sunt obţinute prin manufacturare sau revânzare). Atribute:

- CodPr - codul produsului;
- DenPr - denumirea;
- UM - unitatea de măsură a produsului;
- Grupa - grupa de mărfuri (sortimente) în care se încadrează; acest atribut este important pentru analiza vânzărilor;
- ProcTVA - procentul TVA care se aplică la preţul de vânzare (preţ care este fară TVA); pare de prisos în condiţiile actuale, când toate produsele au un singur procent, 19%, dar nimeni nu poate garanta cum vor gândi promoţiile viitoare de guvernanţi „relaxarea fiscală";

PRODUSE

CodPr	DenPr	UM	Grupa	ProcTVA
1	Produs 1	buc	Tigari	0.19
2	Produs 2	kg	Bere	0.19
3	Produs 3	kg	Bere	0.19
4	Produs 4	l	Dulciuri	0.19
5	Produs 5	buc	Tigari	0.19

Figura 1.18. Tabela PRODUSE

Cheia primară este atributul CodPr. Alte restricţii-utilizator:

- DenPr şi Grupa încep cu majusculă; restul literelor sunt mici;
- unitatea de măsură (UM) se scrie numai cu litere mici.

Tabela pare una „inofensivă", dar de modul său de organizare depinde rezolvarea unor probleme deosebit de sensibile. Este foarte important de stabilit regimul în care se va lucra cu produsele care se comercializează sub mai multe forme. După cum se observă, atributul DenPr nu a fost declarat cheie alternativă. De ce? Să luăm un exemplu absolut la întâmplare: o firmă de comerţ en-gros vinde (legal), printre altele, produsul *Vodcă Scandic* şi la cutii de 1 litru, şi la cutii de 250 ml.

Deşi este acelaşi produs (din prea numeroasele mele încercări, gustul este acelaşi - sau cel puţin aşa îmi amintesc -, indiferent de tipul cutiei), pentru o evidenţă corectă a vânzărilor, este necesar ca în tabela PRODUSE să existe două linii afectate vodcii Scandic, fie ca în figura 1.19, fie ca în figura 1.20.

67	Vodca Scandic - cutie 1 l	buc	Spirtoase	0.22
68	Vodca Scandic - cutie 200 ml	buc	Spirtoase	0.22

Figura 1.19. Specificarea în DenPr a tipului cutiei

67	Vodca Scandic	cutie 1 l	Spirtoase	0.22
68	Vodca Scandic	cutie 200 ml	Spirtoase	0.22

Figura 1.20. Diferenţierea (în afară de cod) numai prin **UM**

Este dificil de spus care variantă este mai bună. Eu, unul, aş înclina pentru cea de-a doua, caz în care atributul DenPr nu este cheie candidată.

Tabela **FACTURI** (figura 1.21) conţine câte o linie pentru fiecare factură emisă, factură ce reflectă o vânzare (către un client). Atribute:

- NrFact - numărul facturii;
- DataFact - data întocmirii facturii;
- CodCl - codul clientului căruia i s-au vândut produsele/serviciile consemnate în factură;
- Obs - observaţii; e folosit relativ rar, pentru a introduce eventuale detalii sau probleme care au apărut în legătură cu o factură.

FACTURI

NrFact	DataFact	CodCl	Obs
1111	01/08/2000	1001	NULL
1112	01/08/2000	1005	Probleme cu transportul
1113	01/08/2000	1002	NULL
1114	01/08/2000	1006	NULL
1115	02/08/2000	1001	NULL
1116	02/08/2000	1007	Preţul propus iniţial a fost modificat
1117	03/38/2000	1001	NULL
1118	04/08/2000	1001	NULL
1119	07/08/2000	1003	NULL
1120	07/08/2000	1001	NULL
1121	07/08/2000	1004	NULL
1122	07/08/2000	1005	NULL

Figura 1.21. Tabela FACTURI

Cheia primară este atributul NrFact. Explicaţia este una simplă: gestionarea facturilor emise este, conform legii, strictă. Nu pot exista două facturi (care reflectă două vânzări) cu un acelaşi număr. Sau, de fapt, pot exista, dar aceasta înseamnă o ilegalitate destul de gravă (se mai poartă încă prin economia subterană). CodCl este cheie străină (tabela-părinte este CLIENŢI). Ca restricţie-utilizator suplimentară, nu pot fi introduse facturi întocmite înainte de **1** august **2000** (DataFact **>= {01/08/2000}).**

Tabela **LINIIFACT** (figura **1.22)** detaliază tabela precedentă. Un tuplu se referă la un produs/serviciu vândut şi consemnat într-o factură emisă. Pentru fiecare factură vor fi atâtea linii câte produse/servicii au fost consemnate la vânzarea respectivă. Atribute:

- NrFact-numărul facturii;
- Linie - numărul liniei din factura respectivă;
- CodPr - codul produsului/serviciului vândut;
- Cantitate - cantitatea vândută;
- PretUnit - preţul unitar (fără TVA) la care s-a făcut vânzarea.

Cheia primară este combinaţia NrFact+Linie. NrFact şi CodPr sunt chei străine. Pentru a determina valoarea de încasat a unei facturi (inclusiv TVA), la valoarea fară TVA pentru fiecare linie (obţinută prin produsul Cantitate 11 PretUnit) trebuie adăugată TVA colectată, obţinută prin aplicarea procentului de TVA al produsului/serviciului (atributul ProcTVA din PRODUSE) la valoarea fară TVA.

Tabela **ÎNCASĂRI** (figura 1.23) reprezintă un nomenclator al încasărilor. Printr-o încasare, un client îşi stinge una sau mai multe obligaţii de plată, adică achită una sau mai multe facturi. Documentul primar pe baza căruia se consemnează încasarea poate fi ordinul de plată, cecul, chitanţa etc. Atributele tabelei sunt:

---

11 Codlnc - codul încasării este un număr intern, util pentru a diferenţia o încasare de celelalte;

- Dat a Inc - data încasării - data la care banii au intrat în contul sau casieria firmei;
- CodDoc - codul documentului justificativ al încasării: OP - ordin de plată, CHIT - chitanţă, CEC - filă cec;
- NrDoc - numărul documentului justificativ;
- DataDoc - data la care a fost întocmit documentul justificativ; din momentul întocmirii documentului de plată până la data la care banii ajung efectiv în cont/casierie trec câteva zile sau săptămâni (datorită circuitului documentelor între firme şi bănci).

Cheia primară este atributul Codlnc. Ca restricţii-utilizator pot fi instituite:

- data încasării nu o poate preceda pe cea a întocmirii documentului (DataDoc <= Datalnc);

L1NIIFACT

NrFact	Linie	CodPr	Cantitate	PretUnit
1111	1	1	50	10000
1111	2	2	75	10500
1111	3	5	500	6500
1112	1	2	80	10300
1112	2	3	40	7500
1113	1	2	100	9750
1114	1	2	70	10700
1114	2	4	30	15800
1114	3	5	700	6400
1115	1	2	150	9250
1116	1	2	125	9300
1117	1	2	100	10000
1117	2	1	100	9500
1118	1	2	30	11000
1118	2	1	150	9300
1119	1	2	35	10900
1119	2	3	40	7000
1119	3	4	50	14000
1119	4	5	750	6300
1120	1	2	80	11200
1121	1	5	550	6400
1121	2	2	100	10500

Figura 1.22. Tabela LINIIFACT

- data documentului de încasare nu poate fi anterioară datei de început a aplicaţiei: 1 august 2000 (DataDoc >= 01-08-2000);
- codul documentului se scrie numai cu majuscule.

INCASARI

CodInc	DataInc	CodDoc	NrDoc	DataDoc
1234	15/08/2000	**OP**	111	10/08/2000
1235	15/08/2000	CHIT	222	15/08/2000
1236	16/08/2000	OP	333	09/08/2000
123/	17/08/2000	CEC	444	10/08/2000
1238	17/08/2000	OP	555	10/08/2000
1239	18/08/2000	OP	666	11/08/2000

Figura 1.23. Tabela INCASARI

Tabela **INCASFACT** (figura **1.24)** detaliază tabela precedentă şi indică ce facturi (tranşe din facturi) sunt achitate prin fiecare încasare. Un client poate plăti mai multe facturi odată (printr-o singură plată). Pe de altă parte, orice factură poate fi plătită în una sau mai multe tranşe, în funcţie de banii de care dispune clientul la momentul întocmirii documentului de plată. Atributele tabelei sunt:

- Codlnc - codul încasării;
- NrFact - factura pentru care se încasează valoarea integrală sau numai o tranşă;
- Transa - tranşa din factură (sau întreaga valoare) care se încasează prin documentul primar ce stă la baza încasării.

INCASFACT

CodInc	NrFact	Transa
1234	1111	539962.5
1234	1118	102637.5
1235	1112	487705
1236	1117	975410
1236	1118	102637.5
1236	1120	731557
1237	1117	975410
1238	1113	1160250
1239	1117	369680

Figura 1.24. Tabela INCASFACT

Cheia primară este combinaţia Codlnc+NrFact, deoarece la o încasare se pot achita mai multe facturi, iar pe de altă parte, o factură poate **fi** plătită în mai multe tranşe. Codlnc şi NrFact sunt chei străine.

Comentarii suplimentare privind restricţiile referenţiale

Instituirea unei restricţii referenţiale într-o bază de date interzice apariţia de valori nenule în tabele-copil care nu se regăsesc în tabelele-părinte (linii orfane). De aceea, la inserarea unei linii într-o tabelă-copil sau modificarea unei chei străine, SGBD-ul trebuie să verifice dacă noile valorile se regăsesc în linii-părinte.

înserarea unei linii într-o tabelă-părinte, ca şi ştergerea de linii dintr-o tabelă-copil nu prezintă nici un pericol „referenţial".

La ştergerea unei linii dintr-o tabelă-părinte trebuie precizat cum se va prezerva restricţia referenţială: fie prin ştergerea în cascadă a tuturor înregistrărilor-copil (ON DELETE CASCADE), fie, pur şi simplu, prin interzicerea ştergerii (ON DELETE RESTRICT).

In fine, la modificarea unei chei primare/alternative pentru care există, în alte tabele, înregistrări-copil trebuie precizată acţiunea întreprinsă de SGBD: modificarea în cascadă a tuturor liniilor-copil (ON UPDATE CASCADE) sau interzicerea modificării, dacă există cel puţin o înregistrare-copil (ON UPDATE RESTRICT).

Pentru exemplificare, să luăm cazul tabelei FACTURI şi trei situaţii legate de actualizarea acesteia.

» *Inserarea* unei linii. Deoarece FACTURI este legată (ca tabelă-copil) prin restricţie referenţială de tabela CLIENŢI (părinte), trebuie verificat dacă valoarea CodCl există în CLIENŢI. Dacă nu, inserarea trebuie

prohibită.

® *Modificarea* unei linii. Aici sunt două situaţii, corespunzătoare posturilor de părinte şi copil a tabelei modificate. Astfel:
- dacă se modifică valoarea atributului CodCl, trebuie verificat dacă aceasta se regăseşte în CLIENŢI şi, dacă nu, operaţiunea trebuie anulată;
- dacă se modifică valoarea lui NrFact, atunci trebuie testat dacă există înregistrări- -copil în tabelele LINIIFACT şi INCASFACT. Dacă da,
  - fie se interzice modificarea (ON UPDATE RESTRICT),
  - fie se modifică în cascadă toate liniile-copil - valorile NrFact din LINIIFACT şi INCASFACT (ON UPDATE CASCADE).

Fireşte, nu e obligatoriu ca acţiunea să fie identică pentru ambele tabele-copil. Cu alte cuvinte, se poate recurge, pentru LINIIFACT, la ON UPDATE CASCADE, iar pentru INCASFACT, la ON UPDATE RESTRICT.

o *Ştergerea* unei linii. Atunci trebuie testat dacă există înregistrări-copil în tabelele LINIIFACT şi INCASFACT. Dacă da,
- fie se interzice ştergerea (ON DELETE RESTRICT),
- fie se şterg în cascadă toate liniile-copil din LINIIFACT şi INCASFACT (ON DELETE CASCADE).

Există diferenţe semnificative între SGBD-uri în privinţa definirii acţiunilor ce trebuie întreprinse pentru respectarea integrităţilor referenţiale. Unele produse - fireşte, dintre cele mai bine cotate - permit, la declararea restricţiilor referenţiale (este de obicei simultană cu crearea tabelelor), instituirea oricărei reguli din cele patru: ON UPDATE CASCADE, ON UPDATE RESTRICT, ON DELETE CASCADE, ON DELETE RESTRICT.

Altele, precum Oracle 8, permit declararea acţiunilor numai pentru ştergere, ON DELETE CASCADE şi ON DELETE RESTRICT. Pentru modificarea în cascadă sunt necesare declanşatoare speciale construite în limbajul de dezvoltare - PL/SQL.

Cât priveşte Visual FoxPro 6, al treilea SGBD pe care îl vom folosi în exemplele din capitolele următoare, este interesant faptul că toate acţiunile pot fi definite, dar numai grafic, prin Referenţial Integrity Builder. în schimb, dezvoltatorii de aplicaţii trebuie să scrie şi să implementeze declanşatoare în care să se descrie explicit ce trebuie întreprins în fiecare situaţie ce pune în pericol restricţiile referenţiale.

1.6. Alte noţiuni ale SGBD-urilor relaţionale

Este greu de inventariat toate sintagmele vehiculate de dezvoltatorii de aplicaţii cu baze de date, cu atât mai mult cu cât unele sunt specifice fiecărui produs. în cele ce urmează ne vom opri, pentru câteva minute, la tabelele derivate (view-urile) şi procedurile stocate.

Tabele virtuale (view-*uri)*

O altă noţiune (alunecoasă) a modelului relaţional este cea căreia, prin traducerea din „originalul" *view* (imagine, vedere), i s-au asociat multiple titulaturi în literatura de specialitate şi practica din ţara noastră: *imagine, relaţie (tabelă) virtuală, relaţie (tabelă) derivată* sau *relaţie (tabelă) dinamică.*

O relaţie virtuală stabileşte o legătură semantică între relaţii statice şi/sau alte relaţii dinamice, nefiind definită explicit, prin tupluri proprii, ca o relaţie de bază (statică), ci printr-o expresie relaţională. Conţinutul (instanţierea) relaţiei virtuale depinde, la un moment oarecare dat, de conţinutul tabelelor de bază din care derivă. Pentru a înţelege mai bine diferenţa, explicaţia se poate rezuma astfel: *tabela virtuală este cea pentru care pe disc se memorează numai schema, nu şi conţinutul.*

Tabelele virtuale oferă oricărui utilizator al unei baze de date posibilitatea prezentării datelor în funcţie de nevoile sale specifice. De asemenea, raţiuni de securitate şi confidenţialitate a anumitor informaţii pot conduce la izolarea unor date faţă de utilizatorii neautorizaţi, lucru deplin posibil prin intermediul *imaginilor*. Pornind de la aceleaşi tabele de bază, se pot crea un mare număr de tabele virtuale, în funcţie de situaţie.

Tabelele derivate sunt suportul creării schemelor externe. O dată definită, o tabelă virtuală poate fi privită ca o tabelă de bază oarecare. De asemenea, ca şi relaţiile de bază, şi „imaginile" pot fi actualizate, ceea ce atrage modificarea tabelelor statice din care derivă. In aceste situaţii apar o serie de probleme. Este clar că modificarea conţinutului tabelelor de bază presupune modificarea conţinutului tabelei (sau tabelelor) derivate. Insă ce informaţii pot fi modificate în tabelele de bază, pornind de la modificările tabelei virtuale? Soluţiile implementate în SGBDR-urile actuale diferă de la caz la caz.

Pentru a intra în câteva detalii, să imaginăm o tabelă derivată denumită ÎNCASĂRI CLIENŢI - figura 1.25 -, ce conţine documentul de încasare, suma încasată (corespunzătoare documentului) şi clientul care a efectuat plata.

INCASARI CLIENŢI

Client	CodDoc	NrDoc	DataDoc	SumaPlata
Client 1 SRL	CEC	444	10-Auc|-	863636
Client 1 SRL	OP	111	10-Auq-2000	5689636
Client 1 SRL	OP	333	9-Aug-2000	2420124
Client 1 SRL	OP	666	11-Aug-2000	327318
Client 2 SA	OP	555	10-Auq-2000	1027295
Client 5 SRL	CHIT	222	15-Aug-2000	431818

Figura 1.25. O tabelă derivată

Utilitatea unei asemenea relaţii virtuale poate fi certificată de un angajat al compartimentului financiar care se ocupă, printre altele, şi cu evidenţa încasărilor de la clienţi. Pe cât de utilă, pe atât de problematică devine ÎNCASĂRICLIENŢI atunci când se pune problema actualizării sale şi propagării modificărilor în tabelele de bază din care provine, CLIENŢI, FACTURI, ÎNCASĂRI şi INCASFACT.

ze Iată câteva dintre probleme:
ne

- modificarea atributului Suma_Plata este imposibil de operat în tabela de bază, INCASFACT. O linie din tabela derivată corespunde uneia sau mai multor linii din tabela de bază (în funcţie de câte facturi sunt achitate prin documentul de plată respectiv). Dacă pentru ordinul de plată 111 din 10 august 2000 se doreşte modificarea (corecţia) sumei din 5.689.636 în 5.600.000, nu se poate cunoaşte tranşa şi factura unde in s-a comis eroarea şi, normal, unde trebuie operată modificarea.

ie
ă)
ci
in
ai
sc

- modificarea celui de-al cincilea tuplu din (Client 2 SA, OP, 555, 10-Aug-2000, 1027295) în (Client 5 SRL, OP, 555, 10-Aug-2000, 1027295), adică modificarea clientului pentru Ordinul de plată 222 poate fi operată în trei moduri în tabele de bază: ţii
  - fie se modifică în FACTURI pentru factura 1113 valoarea atributului CodCl din 1002 în 1005;
  - fie se modifică în linia din INCASFACT codul încasării din 1238 în 1235;
  - fie se modifică în INCASFACT valoarea atributului NrFact din 1113 în 1112, păstrând neschimbat codul încasării.

Chiar dacă nu la fel de plauzibile, cele trei variante generează o „stare" de confuzie j pentru SGBD.

-ii
le

- inserarea unei linii în tabela derivată este o acţiune temerară, dar fără prea mulţi sorţi de izbândă: oricare ar fi cele trei tabele de bază unde are loc inserarea, cel puţin un atribut important (ce nu poate avea valori NULL) nu are valori specificate, astfel încât se _ încalcă una dintre restricţii (de entitate sau comportament). Spre exemplu, convenim că inserarea trebuie să se propage numai în INCASFACT. Nu se cunoaşte însă numărul facturii încasate. Cum NrFact este un component al cheii primare a tabelei, este clar j că operaţiunea va fi interzisă de SGBD.

..
,
le

- ştergerea unei linii din tabela derivată ridică problema identificării liniei sau liniilor din una sau mai multe tabele de bază.

Păstrarea integrităţii BD în momentul actualizării unei tabele derivate se poate rezolva, tă în principiu, prin „apelarea" la una dintre următoarele trei soluţii[27]:

•[a]
- Actualizarea unei relaţii dinamice se face exclusiv pornind de la tabelele de bază din care derivă. Este cea mai restrictivă şi în contradicţie cu regulile ce definesc un SGBDR, aşa cum au fost ele definite de Codd.
- Definirea unor proceduri generale de corespondenţă, pomindu-se de la expresia relaţională de creare a tabelei derivate, proceduri pe baza cărora SGBD-ul va „traduce" actualizarea tabelei virtuale în modificări ale tabelelor de bază.
- Controlul actualizării relaţiei derivate prin intermediul unuisub-sistem de integritate al bazei de date. Această soluţie este considerată ca fiind cea mai eficace pentru păstrarea coerenţei şi securităţii întregului ansamblu de relaţii ce alcătuiesc baza de date, fie ele statice sau dinamice.

Pe de altă parte, nu toate tabelele derivate sunt atât de problematice precum cea discutată anterior, astfel încât propagarea modificărilor dintr-o relaţie virtuală poate fi destul de simplă. După cum vom vedea în capitolele viitoare, restricţiile la care trebuie să se supună o tabelă derivată petru a putea fi modificabilă diferă de la un SGBD la altul.

Câteva dintre SGBDR-urile actuale propun un alt mecanism oarecum apropiat de cel al tabelei virtuale, anume cel de **clişeu** sau **imagine concretă**[12]**.** Acestă noţiune este fondată pe conţinutul unei „imagini" calculate pe baza definiţiei sale şi stocată în baza de date.

O „imagine concretă" reflectă starea bazei de date la un moment *t* când a fost „calculată", clişeul fiind deci determinat periodic de către sistem. Actualizarea bazei de date va fi reflectată în clişeu abia în momentul primului calcul de după aceasta. Firește, la un volum mare al bazei de date, timpul de calcul poate fi destul de lung. De aceea, se poate institui un mecanism de schimbare în clişeu numai a tuplurilor ce au suferit modificări după calculul precedent. Probleme mai delicate apar la ştergerea unor tupluri în tabelele de bază, deoarece un tuplu din clişeu poate proveni din mai multe relaţii, iar ştergerea unuia într-o astfel de relaţie nu trebuie să atragă neapărat ştergerea din clişeu.

Proceduri stocate

O *procedură stocată* este o secvenţă de program (cod) care face parte integrantă din baza de date. Avantajele utilizării procedurilor stocate decurg din faptul că acestea sunt parte din structura (schema) bazei, fiind păstrate în dicţionarul de date (catalogul sistem). Există mai multe tipuri de proceduri stocate: funcţii-utilizator pentru validarea tabelelor, funcţii pentru calculul unor valori implicite, proceduri/funcţii de validare la nivel de linie sau tabelă, funcţii/proceduri de calcul a unor expresii complexe etc.

Declanşatorul (trigger) este un tip special de procedură stocată care este executată automat când un eveniment predefinit (inserare, actualizare sau ştergere) modifică o tabelă. Utilitatea declanşatoarelor este evidentă la formularea unor restricţii mai complexe decât „suportă" comenzile CREATE /ALTER TABLE, actualizarea automată a unor atribute calculate, jumalizarea modificărilor suferite de baza de date, păstrarea integrităţii referenţiale etc.

Există diferenţe sensibile între SGBD-uri şi în ceea ce priveşte sintaxa declanşatoarelor, deoarece acestea sunt redactate în extensiile procedurale ale SQL care prezintă diferenţieri majore de la un producător la altul. Poate fi evocată, spre exemplu, diversitatea declanşatoarelor în Oracle, DB2 etc. faţă de Visual FoxPro.

Astfel, dacă în VFP există numai trei tipuri de declanşatoare, pentru adăugare, pentru modificare şi pentru ştergere, în schimb, în Oracle, pentru fiecare din cele trei operaţiuni există o primă separare între declanşatoare lansate *înaintea* operaţiei (actualizării) şi cele după *operaţie.* La această delimitare se mai adaugă şi cea dintre *triggere* la nivel de linie, ce intră în acţiune la fiecare inserare/modificare/ştergere a unei linii, şi cele la nivel de comandă, care se execută o singură dată, indiferent câte linii afectează comanda de actualizare respectivă (*row* versus *statement).*

Detalii despre declanşatoare în Visual FoxPro şi Oracle vor fi prezentate în capitolul 8.

---

12 Vezi [Saleh94], p. 64.

1.7. Regulile modelului relaţional

în 1985, Codd a publicat în revista *Computerworld* un articol în care formulează 12 reguli, plus regula 0, de fundament, care ar trebui respectate de un SGBD pentru a fi declarat relaţional[13]. Mai mult, autorul s-a angajat cu înverşunare împotriva celor care au folosit abuziv sintagma *relaţional* pentru produsele lor, chiar două firme fiind aduse în pragul falimentului de atacurile „demascatoare" ale lui Codd, sau cel puţin aşa spune legenda[14].

Discutabile, ca orice reguli, „cele 13 porunci" constituie un etalon plauzibil de a caracteriza funcţionalităţile unui SGBD şi a-1 încadra pe scala relaţională.

1. *Regula de fundament.* Orice sistem declarat relaţional trebuie să fie capabil să administreze întreaga bază de date exclusiv prin funcţiuni relaţionale.
2. *Regula informaţiei.* Toate informaţiile sunt reprezentate explicit, la nivel logic, într-un singur mod, ca valori în anumite tabele.
3. *Regula accesului garantat.* Orice dată elementară (reprezentată printr-o valoare atomică) este accesibilă, fară ambiguitate, prin cunoaşterea numelui tabelei din care face parte, a denumirii atributului care o reprezintă şi a valorii cheii primare a tuplului pe care se găseşte.
4. *Prelucrarea sistematică a valorilor nule.* Un SGBDR utilizează valorile nule (diferite de valorile vide, spaţii sau 0) în vederea reprezentării informaţiilor care lipsesc, necunoscute sau inaplicabile. Poate avea valori NULLe orice tip de atribut.
5. *Regula catalogului actualizabil în timp real.* Descrierea unei BDR este stocată, la nivel logic, într-un catalog (dicţionar de date) alcătuit din tabele-sistem ce pot fi consultate de o manieră similară tabelelor de date obişnuite.
6. *Regula sub-limbajului de date* se referă la existenţa cel puţin a unui limbaj (ca SQL, de exemplu) „dotat" cu o serie de comenzi cu sintaxă bine definită, prin care să se realizeze:
   - definirea (descrierea) datelor;
   ® definirea sub-schemelor („machete" sau tabele virtuale);
   - manipularea datelor (interactiv sau prin program);
   ® definirea şi implementarea restricţiilor de integritate;
   ® autorizarea accesului la date;
   ® gestiunea tranzacţiilor.

   Limbajul trebuie să prezinte o sintaxă liniară şi să poată fi utilizat atât interactiv, cât şi din cadrul aplicaţiilor.
7. *Regula actualizării tabelelor virtuale.* Orice tabelă derivată teoretic actualizabilă trebuie să aibă acelaşi regim în cadrul SGBD-ului.
8. *Inserarea, modificarea şi ştergerea pot fi efectuate prin intermediul unui limbaj de nivel înalt, orientat pe lucrul simultan cu un ansamblu de linii.* Ca efect, o tabelă de bază sau derivată poate fi operand nu numai în interogări, dar şi în operaţiuni de inserare/ modificare/ştergere.
9. *Independenţa fizică a datelor.* Aplicaţiile-program şi activităţile de tip „consolă" (interogări directe) rămân neschimbate, din punct de vedere logic, la modificarea în structurile de stocare şi metodele de acces.
10. *Independenţa logică a datelor.* Această regulă permite modificarea dinamică a modelului logic al

---

13 O prezentare a celor 13 reguli se găseşte în lucrarea [Pascu&Pascu94], pp. 22-27. Vezi şi [Lungu ş.a. 95], pp. 133-135.

14 Vezi şi *Relaţional Philosopher,* în [Celko99], pp. 34-35.

bazei de date, cum ar fi, spre exemplu, o descompunere (spargere) sau joncţionare de tabele care nu atrage pierderi de informaţii.

11. *Independenţa mecanismului de integritate al bazei.* Restricţiile de integritate sunt definite printr-un limbaj relaţional şi stocate în dicţionarul de date (catalog), şi nu în aplicaţiile ce exploatează BD. Dintre aceste restricţii trebuie să fie implementabile măcar două, cea a entităţii şi cea referenţială.
12. *Regula independenţei distribuirii datelor.* Limbajul relaţional operează asupra datelor care sunt plasate atât pe calculatorul pe care se derulează activităţile, dar şi pe alte calculatoare legate în reţea, de o manieră transparentă pentru utilizator.
13. *Regula non-subversiunii.* Dacă un sistem relaţional prezintă un limbaj de nivel scăzut (apropiat de „maşină"), care permite prelucrarea, pe rând, a fiecărei înregistrări (linii sau tuplu), acest limbaj nu poate fi utilizat pentru a încălca restricţiile declarate printr-un alt limbaj, de nivel înalt, al SGBD.

Cum găsirea unor SGBD-uri care să îndeplinească toate „poruncile" se dovedeşte o muncă aproape zadarnică (nu cunosc nici un SGBD în totală conformitate cu toate cele 13 reguli), s-a încercat formularea unor *cerinţe minimale* pentru care un sistem de gestiune a bazelor de date să fie declarat „relaţional". Acestea ar fi:

© Toate datele bazei sunt organizate în relaţii.
© între tabele nu există pointeri care să fie gestionaţi de utilizatori.
© Sunt implementaţi operatorii relaţionali: selecţie, proiecţie şi joncţiune naturală.

Un sistem este *complet relaţional* dacă, în plus:

© Sunt implementaţi toţi operatorii algebrici relaţionali.
© Sunt respectate restricţiile de unicitate a cheii şi cea referenţială.

Sistemele care îndeplinesc numai condiţiile © şi © sunt *pseudorelaţionale,* iar dintre cele pseudorelaţionale, cele care îndeplinesc © numai în raport cu funcţia de interogare sunt denumite SGBD-uri cu *interfaţă relaţională.*

în afara celor 13 porunci, Codd a mai specificat 9 reguli structurale, 3 reguli de intregritate şi 18 reguli de manipulare. Ulterior, a transformat poruncile în veritabilă Constituţie, formulând, în a doua versiune a modelului relaţional, 333 de reguli. Dacă se mai străduieşte un pic şi mai extrage încă un rând de specificaţii, obţinem 666 de reguli, iar atunci se poate spune că modelul relaţional este unul dumnezeiesc.

Capitolul 2

ALGEBRA RELAŢIONALĂ

în primul capitol am văzut că modelul relaţional formulat de Codd are la bază trei elemente: structuri, operaţii şi reguli de integritate. Pentru exprimarea operaţiilor aplicabile structurilor de date, autorul a definit un limbaj de manipulare a datelor puternic matematizat, bazat pe teoria ansamblurilor (seturilor), numit DSL/Alpha[15]. Prezentul capitol este dedicat unui limbaj teoretic - algebra relaţională - care poate constitui un bun punct de plecare în înţelegerea chestiunilor esenţiale ale celui mai important limbaj dedicat bazelor de date - SQL.

2.1. Caracterizare generală a limbajelor de interogare

Gestiunea bazelor de date relaţionale are ca obiectiv principal acoperirea nevoilor informaţionale ale conducerii firmei la toate nivelurile. Până la consacrarea SGBDR-urilor, extragerea informaţiilor dorite din baza de date se realiza prin aplicaţii dezvoltate exclusiv cu limbaj*e procedurale,* în care se precizează atât datele dorite, cât şi metodele de căutare şi extragere a acestora. Actuala generalizare a SGBDR-urilor se află într-o strânsă relaţie cu elaborarea şi implementarea unor limbaje performante pentru manipularea BD - *limbajele de interogare.*

Limbajele relaţionale sunt *neprocedurale:* utilizatorul defineşte datele ce trebuie extrase din BD, sarcina căutării şi extragerii fiind „rezervată" exclusiv SGBD-ului. De asemenea, în unele lucrări, acestea sunt considerate *limbaje închise,* deoarece o consultare generează o nouă relaţie ce poate fi utilizată, la rândul său, ca argument în alte consultări ş.a.m.d.

Pornind de la cele două modalităţi de definire a relaţiei, pe de o parte, ca predicat aplicat asupra unor domenii şi, pe de altă parte, ca ansamblu de tupluri, limbajele de manipulare a datelor sunt grupate în două mari categorii: *limbaje predicative* - fondate pe teoria predicatelor şi *limbaje asambliste* — fondate pe teoria ansamblurilor (tuplurilor).

Limbajele *predicative* sunt divizate în alte două sub-clase:

- cele care au la bază *calculul relaţional asupra tuplurilor* (limbaje orientate pe tupluri) şi
- cele în care *calculul relaţional se aplică asupra domeniilor* (limbaje orientate pe domenii).

15 Vezi şi [Date99-1],

Elementul definitoriu pentru limbajele de manipulare bazate pe calculul predicatelor îl reprezintă noţiunea de *variabilă,* care poate fi asociată fie tuplurilor, fie domeniilor.

O altă grupare delimitează limbajele non-grafice de cele grafice. Primele permit reprezentarea unei consultări „în linie", prin dispunerea succesivă a operatorilor, atributelor şi relaţiilor. Cele grafice permit redactarea consultării în mod interactiv, prin afişarea pe ecran a unui sistem de meniuri şi elemente de dialog din care opţiunile pot fi selectate şi modificate cu ajutorul mouse-ului (sau claviaturii); în plus, se mai poate opera o delimitare şi pentru limbajele grafice în funcţie de utilizarea *explicită* sau *implicită* a variabilelor de domeniu.

Deşi nu exagerat de recentă, o clasificare încă valabilă (în parte) a limbajelor relaţionale poate fi prezentată ca în figura *2A*[32].

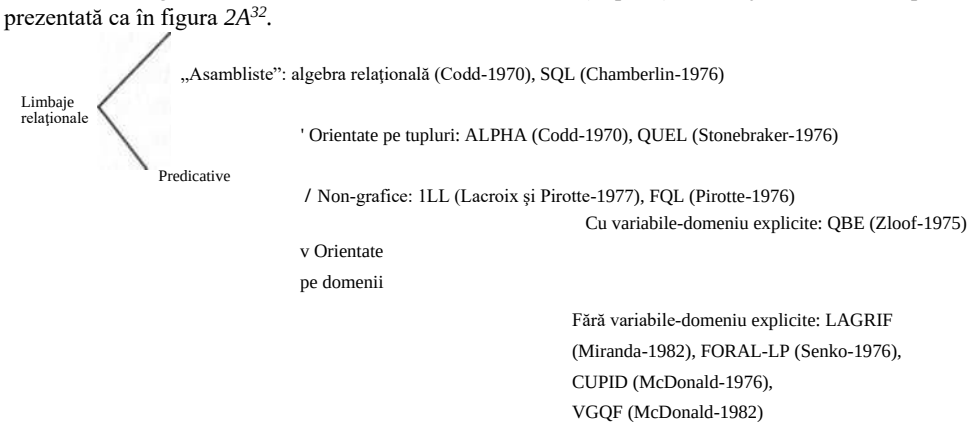

Limbaje relaţionale
„Asambliste": algebra relaţională (Codd-1970), SQL (Chamberlin-1976)
Predicative
' Orientate pe tupluri: ALPHA (Codd-1970), QUEL (Stonebraker-1976)
/ Non-grafice: 1LL (Lacroix şi Pirotte-1977), FQL (Pirotte-1976)
Cu variabile-domeniu explicite: QBE (Zloof-1975)
v Orientate pe domenii
Fără variabile-domeniu explicite: LAGRIF (Miranda-1982), FORAL-LP (Senko-1976), CUPID (McDonald-1976), VGQF (McDonald-1982)

Figura 2.1. O clasificare a limbajelor relaţionale

Există o serie de caracteristici comune tuturor limbajelor:

- *operatorii relaţionali se aplică relaţiilor luate în întregime, adică tuturor tuplurilor care alcătuiesc relaţiile respective',*
- *rezultatul fiecărui operator* (rezultatul consultării) *este o nouă relaţie* ce poate servi ca argument într-o altă consultare ş.a.m.d.;
- *logica operatorilor se bazează pe valorile atributelor, ceea ce constituie, de altminteri, suportul singurului mod de acces în BD.* întrucât consultările mono- şi multi-relaţii sunt efectuate exclusiv prin compararea valorilor atributelor (definite pe domenii compatibile), accesul total independent de limbaj este asigurat.

înaintea redactării unei consultări într-un limbaj relaţional trebuie parcursă o fază de *analiză,* pentru determinarea atributelor rezultatului, legăturilor dintre tabele şi eventualelor condiţii (restricţii) ce trebuie respectate.

Limbajul algebric relaţional cuprinde două tipuri de operatori: *asamblişti* - REUNIUNE, INTERSECŢIE, DIFERENŢĂ, PRODUS CARTEZIAN - şi *relaţionali* - SELECŢIE, PROIECŢIE, JONCŢIUNE şi DIVIZIUNE.

O altă clasificare distinge operatorii fundamentali, ireductibili (reuniunea, diferenţa, produsul cartezian, selecţia, proiecţia), de cei derivaţi, a căror funcţionalitate poate fi realizată prin combinarea operatorilor fundamentali (intersecţia, joncţiunea, diviziunea).

33. [Saleh94], p. 35.

Pentru cele ce urmează se notează cu:

- *t* sau *r*, un *tuplu* al unei relaţii (linie a unei tabele) şi
- *t(A),* un *sub-tuplu* al relaţiei R, relativ la atributul A (valoarea atributului A în linia *t).*

Algebra, ca şi calculul relaţional, serveşte ca punct de referinţă în caracterizarea unui limbaj ca fiind *complet* sau incomplet, din punct de vedere relaţional. Dacă un limbaj permite exprimarea tuturor operatorilor enumeraţi mai sus şi oferă cel puţin facilităţile algebrei relaţionale, se spune că acesta este un limbaj relaţional complet[33].

Mai trebuie amintit că în literatura de specialitate există o multitudine de reprezentări ai operatorilor algebrei relaţionale. Notaţia pe care o vom utiliza în cele ce urmează este cea mai simplă şi uşor de înţeles *{părerea mea,* vorba lui...).

2.2. Operatorii asamblişti

Trei dintre operatorii asamblişti - *reuniunea* („u"), *intersecţia* („n") şi *diferenţa* („-") - pot opera numai cu două relaţii *unicompatibile.*

Fie R1 (Al, A2, . . ., An) şi R2 (Bl, B2, . . ., Bm) două relaţii. Se spune despre R1 şi R2 că sunt unicompatibile dacă:

1. n = m
2. V i e {1,2, ..., n}, Ai şi Bi sunt de acelaşi tip sintactic (aceasta nu înseamnă că trebuie să prezinte, neapărat, domenii identice de definire).

Relaţiile R1 şi R2 din figura 2.2 sunt unicompatibile deoarece:

1. ambele au acelaşi număr de atribute;
2. atributele A, B, C din R1 (le putem nota şi Rl.A, R1. B, Rl.C) corespund sintactic (sunt de acelaşi tip) atributelor C, D şi E din R2 (R2 . C, R2 . D, R2 . E).

R1

A	B	C
20	XYZ	30
30	XXZ	20
40	YYX	25

R2

C	D	E
25	XYZ	30
40	YYX	25
30	XXZ	40

Figura 2.2. Două relaţii unicompatibile

Reuniunea a două relaţii unicompatibile, R1 şi R2 , este definită astfel: R1 u R2 = {tuplu 111 e R1 sau t e R2 }.

Se notează:

$$R3 \leftarrow R1 \cup R2.$$

32. [Miranda&Busta90], voi. 2, p. 30.

Conţinutul tabelei-reuniune R3 este prezentat în figura 2.3. Primele trei tupluri din rezultat sunt preluate din Rl, iar ultimele două din R2. R3 are numai cinci tupluri deoarece un tuplu este comun tabelelor Rl şi R2. Algebra relaţională elimină automat dublurile (tuplurile identice), astfel încât restricţia de unicitate este asigurată după orice operaţie.

R1

R2

A	B	C	D	E
20	XYZ	30	YZ	30
30	XXZ	20	YX	25
40	YYX	25	XZ	40

R1UR2

A	B	c
20	XYZ	30
30	XXZ	20
40	YYX	25
25	XYZ	30
30	XXZ	40

Figura 2.3. Reuniunea a două relaţii

Există suficiente situaţii informaţionale care fac uz de reuniunea a două tabele. Să luăm două exemple:

- tabelele ce reflectă tranzacţii economice pot fi descompuse (sparte) în funcţie de anul (luna sau cincinalul, deceniul) la care se referă; dacă ne raportăm la baza noastră de date, tabela LINIIFACT poate fi ruptă în alte două: LF_2000_2001, care ar conţine facturile emise în anii 2000 şi 2001, şi LINIIFACT, ce conţine numai înregistrări aferente anului calendaristic 2002. începând cu 2002, orice situaţie statistică privind vânzările în perioada 2000-2002, 2000-2003 etc. necesită reuniunea celor două tabele, LF_2000_2001 şi LINIIFACT.
- pentru a afla care sunt clienţii care au cumpărat cel puţin unul din produsele *Produs 1* şi *Produs 2,* se poate proceda la reuniunea tabelei ce conţine clienţii care au cumpărat *Produs 1* cu tabela clienţilor care au cumpărat *Produs 2.*

Reuniunea este comutativă. Singura problemă neclară ar fi legată de numele atributelor în relaţia-rezultat. în acest sens, se poate institui regula potrivit căreia numele atributelor relaţiei-reuniune sunt numele primei relaţii participante în operaţie. Aceasta nu are importanţă asupra comutativităţii, deoarece conţinutul tabelei-rezultat este identic, indiferent care este prima relaţie enumerată.

Intersectia

t

Intersecţia a două relaţii unicompatibile, Rl şi R2, poate fi definită astfel:

$$Rl \cap R2 = \{\text{tuplu } t \mid t \in Rl \text{ şi } t \in R2\}$$

Se notează:

$$R4 \leftarrow Rl \cap R2.$$

Conţinutul tabelei-intersecţie R4 este prezentat în figura 2.4. Cum numai un tuplu este absolut identic şi în Rl şi R2, tabela-rezultat este alcătuită dintr-o singură linie.

Rl

A	B	C
20	XYZ	30
30	XXZ	20
40	YYX	25

R2

C	D	E
25	XYZ	30
40	YYX	25
30	XXZ	40

Rl nR2

A	B	C
40	YYX	25

Figura 2.4. Intersecţia a două relaţii Exemple de informaţii care fac

necesară recurgerea la intersecţie:

- pentru a afla care sunt clienţii care au cumpărat şi *Produs 1,* şi *Produs 2,* se poate proceda la intersecţia tabelei clienţilor care au cumpărat *Produs 1* cu tabela alcătuită din clienţii care au cumpărat *Produs 2;*
- zilele în care s-au făcut vânzări şi clientului *Client 1 SRL,* şi clientului *Client 2 SA;*
- persoanele de la firmele-client care cumulează posturile de *Director vânzări* şi *Şef aprovizionare.*

Ca şi reuniunea, intersecţia este comutativă, iar numele atributelor relaţiei-intersecţie sunt extrase din prima relaţie participantă în operaţie.

Diferenţa

Diferenţa a două relaţii unicompatibile, notate Rl şi R2, este definită astfel:

$$Rl - R2 = \{\text{tuplu } t \mid t \text{ e } Rl \text{ şi } t \text{ g } R2 \}$$

Se notează:

$$R5 \text{ <- } Rl \text{ n } R2.$$

Conţinutul tabelei-diferenţă R5 (figura 2.5) conţine numai tuplurile din prima relaţie, Rl, care nu se regăsesc în a doua relaţie, R2. Aşadar, din rezultat este eliminat al treilea tuplu din Rl, deoarece valorile acestuia există şi în R2 (al doilea tuplu din R2).

Exemple de informaţii care fac necesară recurgerea la diferenţă:

- care sunt clienţii care au cumpărat *Produs 1* dar nu au cumpărat *Produs 21*
- care sunt zilele în care s-au făcut vânzări clientului *Client 1 SRL,* dar nu există nici o factură către *Client 2 SA?*

Spre deosebire de reuniune şi intersecţie, diferenţa nu este este comutativă. Atributele relaţiei-diferenţă sunt cele ale primei relaţii (descăzutul), iar tuplurile care sunt extrase din relaţia-descăzut nu se regăsesc în relaţia-scâzător. în plus, nu există restricţii privind cardinalitatea (numărul de tupluri) celor două relaţii, adică nu este musai ca relaţia-descăzut să conţină mai multe tupluri decât cea scăzător.

R1

A	B	C
20	XYZ	30
30	XXZ	20
40	YYX	25

R2

C	D	E
25	XYZ	30
40	YYX	25
30	XXZ	40

Rl -R2

A	B	c
20	XY	30
30	lxxz	20

Figura 2.5. Diferenţa a două relaţii

Produsul cartezian

Produsul cartezian dintre două relaţii Rl şi R2, denumit de Codd joncţiune încrucişată (CROSS JOIN), este ansamblul tuturor tuplurilor obţinute prin concatenarea fiecărei linii din tabela Rl cu toate liniile tabelei R2. Formal, dacă notăm cele două relaţii: Rl (Al, A2,
. . ., An) şi R2 (Bl, B2, . . ., Bm), produsul cartezian este definit astfel:

$$Rl \circledR R2 = \{ (t_1, t_2)! \text{ ti e Rl şi } t_2 \, e \, R2 \}$$

Spre deosebire de celelalte trei operaţiuni precedente, produsul cartezian nu face apel la noţiunea de relaţii unicompatibile, iar relaţia-rezultat cumulează atributele celor două relaţii-argument. în figura 2.6 este ilustrat rezultatul produsului cartezian a tabelelor Rl şi R2.

R1 R2

A	B	C	D	E
20	XYZ	30	XYZ	30
30	XXZ	20	YYX	25
40	YYX	25	XXZ	40

Figura 2.6. Produsul cartezian

Se notează:

R6 ← R1 ⊗ R2

Tabela-rezultat R6 are o nouă structură - şase atribute (trei preluate din Rl şi trei din R2). întrucât există un atribut cu nume comun, C, pentru a diferenţia cele două apariţii, acestea sunt prefixate, în antetul tabelei, cu numele relaţiei din care provine.

Prima linie din R6 este obţinută prin „lipirea" primului tuplu din Rl cu primul tuplu din R2, a doua din primul tuplu din Rl cu al doilea din R2 etc. Cum Rl are 3 tupluri, iar R2 tot 3, relaţia-rezultat al produsului cartezian are 3 16 3 = 9 tupluri.

Nu prea există situaţii care să reclame folosirea directă şi exclusivă a produsului cartezian. Cel mai important „merit" al acestuia în algebra relaţională este că permite „alipirea" a două relaţii, fundamentând astfel operatorul-cheie care este joncţiunea.

2.3. Operatorii relaţionali

Cei patru operatori prezentaţi în paragraful precedent sunt generali, spre deosebire de următorii, care sunt specifici algebrei relaţionale. De obicei, gruparea operatorilor relaţionali se face astfel:

- *operatori unari de restricţie,* care permit decupajul unei relaţii, pe orizontală - SELECŢIA şi pe verticală - PROIECŢIA;
- *operatori binari de extensie*: JONCŢIUNEA şi DIVIZIUNEA.

O altă deosebire majoră faţă de paragraful precedent este că vom putea recurge şi la exemple concrete din baza de date „cobai" prezentată în capitolul anterior.

Selecţia

Selecţia tnjuă_xlmt£-o_ relaţie (tabelă) numai tuplurile ce satisfac o condiţie specificată printr-un predicat.

Ca preambul, definim noţiunea de *formulă F asupra unei relaţii R* ca o expresie logică compusă din:

---

16 *operanzi* care sunt nume de atribute sau constante;

- *operatori de comparaţie aritmetică*: >, >, <, <, =,
- *operatori logici*: Şl, SAU, NON.

Selecţia unei relaţii R, printr-o condiţie F, notată SF(R), poate fi definită:

SF(R) = { tuplu t | t s R şi F(t) = adevărată}.

O notaţie ceva mai pământească este:

Rl <—SELECŢIE (R; <expresie-logică>), dar, de dragul

ştiinţei, vom detalia:

- R este relaţia R (Al, A2, . . . An) asupra căreia se aplică selecţia (Ai sunt atributele sale).

- Rl este noua relaţie obţinută în urma selecţiei, care va avea aceeaşi schemă relaţională cu R- Rl (Al, A2, . . . An).

O <expresie-logică> poate fi scrisă mai analitic astfel:

■=> <expresie-logică> = (termeni) şi/sau (termen2)... şi/sau (termenk), unde O termen

j = *expresiei* 0 *expresie*$_2$.

^ *expresie*, sau *expresie*$_2$ sunt expresii calculate plecând de la atributele Ai ale relaţiei R.

=> 0 poate fi unul dintre operatorii pentru comparaţie.

Exemplul 1

Pentru a pune în operă savanta notaţie de mai sus, luăm în discuţie o primă problemă: *Care sunt liniile din Rl pentru care valorile atributelor A şi C sunt mai mari decât 20?* Ajungem, astfel, la notaţia:

R <-SELECŢIE (Rl; A > 20 AND C > 20)

Tabela R este prezentată în figura 2.7.

R

A	B	C
40	YYX	25

Figura 2.7. Rezultat-selecţie - exemplul 1

Exemplul 2

Care sunt judeţele din Moldova?

Mai întâi se identifică în baza de date tabela (sau tabelele) din care se extrage rezultatul. în acest exemplu, aceasta este JUDEŢE. Apoi se stabilesc atributele (atributul) asupra cărora se va aplica predicatul de selecţie. Se obţine soluţia:

R <—SELECŢIE (JUDEŢE; Regiune = "Moldova")

R

Jud	Judeţ	Reqiune
IS	Iasi	Moldova
VN	Vrancea	Moldova
NT	Neamţ	Moldova
SV	Suceava	Moldova
VS	Vaslui	Moldova

Figura 2.8. Rezultat-selecţie - exemplul 2

Exemplul 3

Care sunt facturile emise în perioada 2-5 august 2000?

Tabela în care va opera operatorul de selecţie este FACTURI. Predicatul de selecţie utilizează atributul DataFact:

R -f- SELECŢIE (FACTURI; DataFact >= 02/08/2000 AND DataFact <= 05/08/2000)

R

NrFact	DataFact	CodCl	Obs
1115	02/08/2000	1001	NULL
1116	02/08/2000	1007	Preţul propus iniţial a fost modificat
1117	03/08/2000	1001	NULL
1118	04/08/2000	1001	NULL

Figura 2.9. Rezultat-selecţie - exemplul 3

Proiecţia

Prin proiecţie, o relaţie poate fi „decupată" pe verticală. Dacă selecţia extrage dintr-o tabelă anumite linii, pe baza condiţiei îndeplinite de valorile unora dintre atribute, *proiecţia* permite selectarea într-o tabelă-rezultat numai a coloanelor (atributelor) dorite dintr-o relaţie.

Formal, fie o relaţie R (Al, A2, . . . An). Proiecţia relaţiei R asupra unui subansamblu alcătuit din atribute proprii este o relaţie care se obţine după parcurgerea a doi paşi:

a) eliminarea dintre Ai a acelor atribute care nu sunt specificate şi

b) suprimarea dublurilor (tuplurile identice).

Se notează:

Rl <- PROIECŢIE (R; Aj, Ak, Ax)

Spre deosebire de R, schema relaţiei Rl este alcătuită numai din atributele indicate: Rl (Aj , Ak, . . ., Ax). Dacă după extragerea coloanelor nu există tupluri (linii) identice, Rl va avea acelaşi număr de linii ca şi relaţia R. în caz contrar, numărul lor va fi mai mic, în funcţie de numărul dublurilor.

Exemplul 4

începem, ca de obicei, cu un exemplu ceva mai arid. *Care sunt valorile combinaţiei atributelor A şi C în relaţia Rl?*

R <*r*- PROIECŢIE (Rl; A, C)

Tabela R are două coloane, A şi C, şi trei linii, ca în figura 2.10.

R

A	C
20	30
30	20
40	25

Figura 2.10. Rezultat-proiecţie - exemplul **4**

Exemplul 5

Ce regiuni ale ţării sunt preluate în bază?

Tabela în care se află răspunsul este JUDEŢE. Singura coloană care interesează este Reqiune (figura 2.11).

R PROIECŢIE (JUDEŢE; Regiune)

In primul pas se face decupajul pe verticală, obţinându-se o relaţie notată R', apoi se elimină dublurile, rezultatul final fiind relaţia R.

R'

Regiune
Moldova
Moldova
Moldova
Moldova
Moldova
Banat

R

Regiune
Moldova
Banat

Figura 2.11. Rezultat-proiecţie - exemplul 5

Exemplul 6

Care sunt: codul, denumirea şi numărul de telefon ale fiecărui client?

Tabela care interesează este CLIENŢI, din care se decupează trei coloane: CodCl, DenCl şi Telefon (figura2.12).

R 4- PROIECŢIE (CLIENŢI; CodCl, DenCl, Telefon)

R

CodCl	DenCl	Telefon
1001	Client 1 SRL	NULL
1002	Client 2 SA	032-212121
1003	Client 3 SRL	035-222222
1004	Client 4	NULL
1005	Client 5 SRL	056-111111
1006	Client 6 SA	NULL
1007	Client 7 SRL	056-121212

Figura 2.12. Rezultat-proiecţie - exemplul 6

înlănţuirea consultărilor

f

După cum aminteam în paragraful introductiv ai capitolului, rezultatul unei consultări este o relaţie (tabelă) nouă. Pe baza acestui fapt, se pot înlănţui două sau mai multe operaţiuni, redactându-se astfel interogări complexe.

Exemplul 7

Care este numărul de telefon al clientul Client 2 SA ?

Soluţia este una foarte simpă. Cu ajutorul selecţiei se decupează din relaţia CLIENŢI numai linia corespunzăoare clientului „incriminat". Se obţine o relaţie nou-nouţă denumită (de noi) Rl. Asupra lui Rl se aplică o proiecţie, deoarece interesează numai numărul de telefon; astfel, R2 conţine răspunsul la problema luată în discuţie (vezi figura 2.13).

Rl <- SELECŢIE (CLIENŢI; DenCl = "Client 2 SA")
R2 <- PROIECŢIE (Rl; Telefon)

R1

CodCl	DenCl	CodFiscal	Adresa	CodPost	Telefon
1002	Client 2 SA	R1002	NULL	6600	032-212121

R2

Telefon
032-212121

Figura 2.13. înlănţuirea unei selecţii cu o proiecţie - exemplul 7

Exemplul 8

Care sunt denumirile şi codurile poştale ale localităţilor (prezente în bază) din judeţele Iaşi (IS) şi Vrancea (VN)?

Tabela „interogată" este LOCALITĂŢI. Pentru a răspunde la întrebarea pe care tot noi am formulat-o, putem alege între următoarele două soluţii:

- Soluţia 1 - figura 2.14

Rl 4- SELECŢIE (LOCALITATI; Jud = "IS" OR Jud = "VN") R2 <— PROIECŢIE (Rl;

Loc, CodPost)

R1

CodPost	Loc	Jud
6600	lasi	IS
5725	Paşcani	IS
5300	Focşani	VN

R2

Loc	CodPost
lasi	6600
Paşcani	5725
Focşani	5300

Figura 2.14. Exemplul 8 - soluţia 1

® Soluţia 2 - figura 2.15

Rl <- SELECŢIE (LOCALITATI; Jud = "IS")

R2 <- PROIECŢIE (Rl; Loc, CodPost)

R3 «- SELECŢIE (LOCALITATI; Jud = "VN")

R4 <*r*- PROIECŢIE (R3; Loc, CodPost)

R5 •*- R2 u R4

Prima soluţie este, prin simplitate, cea mai tentantă. Este însă un prim caz în care pentru rezolvarea unei probleme pot fi formulate două (sau mai multe) soluţii.

Exemplul 9

Care sunt codurile produselor care apar deopotrivă în factura 1111 şi în factura 1117?

R1

CodPost	Loc	Jud
6600	lasi	IS
5725	Paşcani	IS

R2

Loc	CodPost
lasi	6600
Paşcani	5725

R3

CodPost	Toc	Jud
5300	Focşani	VN /

X R4

Loc	CodPost
Focşani	5300

R5

Loc	CodPost
lasi	6600
Paşcani	5725
Focşani	5300

Figura 2.15. Exemplul 8 - soluţia 2

Tabela din care vor fi extrase datele este LINIIFACT. Soluţia se bazează pe intersecţia relaţiei care conţine produsele prezente în factura 1111 (R2) cu relaţia produselor prezente în factura 1117 (R4), după cum reiese din figura 2.16.

```
Rl <- SELECŢIE (LINIIFACT; NrFact = 1111)
```

R2 <- PROIECŢIE (Rl; CodPr)

R3 <- SELECŢIE (LINIIFACT; NrFact = 1117) R4 <- PROIECŢIE (R3; CodPr)

R5 <- R2 n R4

R1

NrFact	Linie	CodPr	Cantitate	PretUnit
1111	1	1	50	10000
1111	2	2	75	10500
1111	3	5	500	6500

R2

CodPr
1
2
5

R3

NrFact	Linie	CodPr	Cantitate	PretUnit
1117	1	2	100	10000
1117	2	1	100	9500

R4

CodPr
2
1

R5

CodPr
1

Figura 2.16. Exemplul 9 - soluţie

Joncţiunea

în paragraful anterior am văzut că produsul cartezian permite fuzionarea a două tabele într-o tabelă mamut ce conţine toate atributele şi liniile obţinute prin combinarea fiecărui tuplu dintr-o relaţie cu fiecare tuplu din cealaltă. Tot atunci ne exprimam regretul sincer că operatorul produs cartezian nu poate fi folosit, de unul singur, în interogări, dar că acest „neajuns" va fi compensat din plin de operatorul derivat *joncţiune.*

Dacă produsul cartezian este o fuziune necondiţionată a două tabele, joncţiunea reprezintă fuziunea a două relaţii *care au o proprietate comună.* Fie două relaţii, notate: Rl (Al, A2, An) şi R2(Bl,B2, . . . , Bp). Fie Ai şi Bj două atribute definite pe acelaşi domeniu şi 0 ansamblul operatorilor de comparaţie: {=, >, >, <, < *}, ce pot fi aplicaţi celor două atribute Ai şi B j.

Joncţiunea relaţiei Rl, prin A/', cu relaţia R2, prin *Bj,* notată Rl (Ai

0 Bj)R2sauRl XleR2

este relaţia ale cărei tupluri sunt obţinute prin concatenarea fiecărui tuplu al relaţiei Rl cu tuplurile relaţiei R2, pentru care este verificată condiţia 0 instituită între Ai şi B j.

$$Rl\ (Ai\ 0\ B\,j)\ R\,2 = \{\ 1\ 1\ 1\ \ e\ R\,l\ <8>\ R2\ şi\ t(Ai)\ 0\ t(Bj)\}$$

Joncţiunea este echivalenta unui produs cartezian urmat de o selecţie. Joncţiunea definită mai sus este cunoscută în lucrările de specialitate ca *theta-joncţiune.* în lucrul cu BDR se utilizează cu precădere *echi-joncţiunea,* ce reprezenta un caz particular al theta-joncţiunii, atunci când 0 este operatorul de egalitate („="). Formal, echi-joncţiunea se defineşte astfel:

$$Rl\ (Ai = B\,j)\ R\,2 = \{\ t\ |\ t\ e\ R\,l\ \ ®\ R2\ şi\ t(Ai) = t\ (\ B\,j\ )\ \}$$

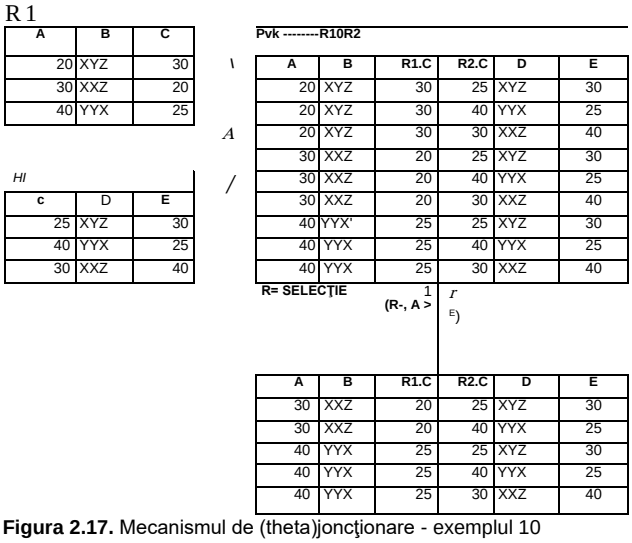


Figura 2.17. Mecanismul de (theta)joncţionare - exemplul 10

Apelăm şi la o altă notaţie, mai uşor de reprezentat şi suficient de inteligibilă.
R <- ECHI-JONCŢIUNE (Rl, R2; Ai=Bj)

:le
ruia
că
;st

Exemplul 10 - Theta-joncţiune
începem exemplificările cu aceleaşi două tabele folosite în precedentul paragraf, Rl Rezultatul şi R2.
joncţiunii (theta-joncţiunii) exprimată prin expresia:

R <— JONCŢIUNE (Rl, R2; Rl.A >= R2.E) va fi obţinut în doi paşi,

după cum este descris în figura 2.17.

ea
te:
ite
ce

Exemplul 11 - Echi-joncţiune

Operatorul de comparaţie dintre cele două atribute este, obligatoriu, semnul de egalitate.

R <- JONCŢIUNE (Rl, R2; Rl.A = R2.E)

CU

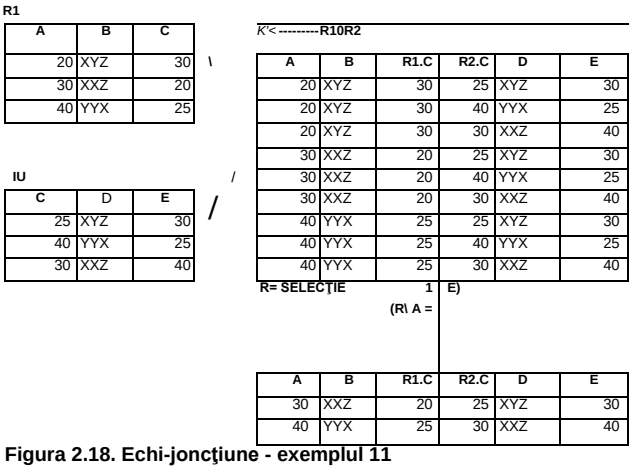

R1
A B C
20 XYZ 30
30 XXZ 20
40 YYX 25
IU
C D E
25 XYZ 30
40 YYX 25
30 XXZ 40
R10R2
A B R1.C R2.C D E
20 XYZ 30 25 XYZ 30
20 XYZ 30 40 YYX 25
20 XYZ 30 30 XXZ 40
30 XXZ 20 25 XYZ 30
30 XXZ 20 40 YYX 25
30 XXZ 20 30 XXZ 40
40 YYX 25 25 XYZ 30
40 YYX 25 40 YYX 25
40 YYX 25 30 XXZ 40
R= SELECŢIE 1
(R\ A =
E)
A B R1.C R2.C D E
30 XXZ 20 25 XYZ 30
40 YYX 25 30 XXZ 40

Figura 2.18. Echi-joncţiune - exemplul 11

Exemplul 12 - Joncţiune naturală

Joncţiunea naturală presupune nu numai ca operatorul de comparaţie să fie semnul de egalitate, ci şi denumirea identică a atributelor de legătură dintre cele două tabele.

R <- JONCŢIUNE (Rl, R2; Rl.C = R2.C)

Datorită faptului că ambele atribute au acelaşi nume, se poate considera cătabela-rezultat păstrează numai unul dintre cele două atribute, ca în figura 2.19, şi se poaterecurge la o notaţie simplificată:

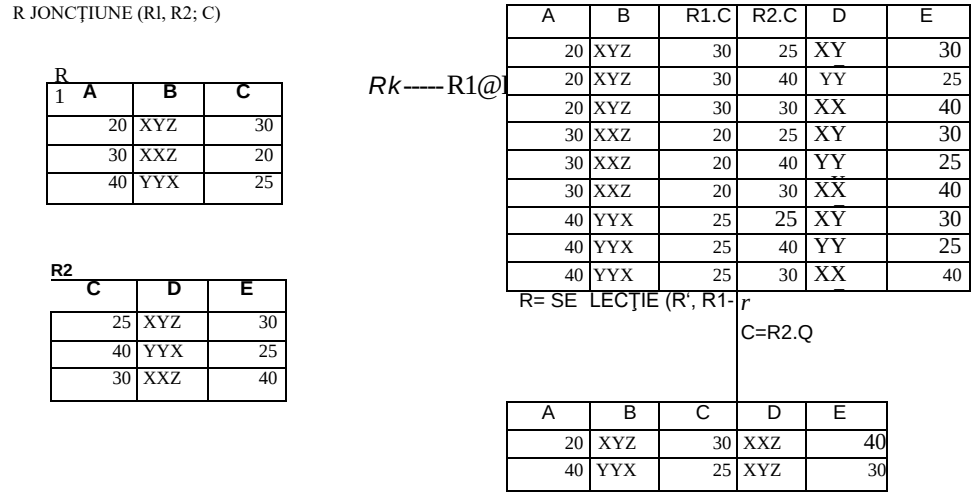

R JONCŢIUNE (Rl, R2; C)
R
1
A B C
20 XYZ 30
30 XXZ 20
40 YYX 25
R2
C D E
25 XYZ 30
40 YYX 25
30 XXZ 40
Rk
A B R1.C R2.C D E
20 XYZ 30 25 XY 30
20 XYZ 30 40 YY 25
20 XYZ 30 30 XX 40
30 XXZ 20 25 XY 30
30 XXZ 20 40 YY 25
30 XXZ 20 30 XX 40
40 YYX 25 25 XY 30
40 YYX 25 40 YY 25
40 YYX 25 30 XX 40
R= SE LECŢIE (R', R1-
C=R2.Q
A B C D E
20 XYZ 30 XXZ 40
40 YYX 25 XYZ 30

Figura 2.19. Joncţiune naturală - exemplul 12

De ce se insistă atât de mult pe importanţa joncţiunii? în primul rând pentru că permite recompunerea relaţiei universale iniţiale. Modelul relaţional se bazează pe spargerea bazei în relaţii, astfel încât nivelul redundanţei datelor şi problemele la actualizarea tabelelor să fie redus la minim. Cele mai multe interogări însă, operează cu date şi predicate aplicate simultan atributelor din două sau mai multe tabele. Cum selecţia este un operator *unar* (poate fi aplicată unei singure relaţii), este necesară fuzionarea prealabilă a celor două, trei... relaţii şi obţinerea unei relaţii-agregat, la care se aplică predicatul suplimentar de selecţie.

Fuzionarea este posibilă prin joncţiune. Prin joncţionarea tuturor relaţiilor dintr-o bază de date se obţine relaţia *universală* (cea iniţială, atotcuprinzătoare). Nu ne permitem să reconstituim structura şi conţinutul relaţiei universale ale bazei de date VANZARI, însă, prin exemple, vom încerca să demonstrăm utilitatea joncţiunii.

Exemplul 13

Să se obţină, pentru fiecare localitate: codul poştal, denumirea, indicativul judeţului, denumirea judeţului şi regiunea din care face parte.

Practic, tabelei LOCALITĂŢI îi trebuie „alipite" la dreapta informaţiile din tabela JUDEŢE Problema este dificil de formulat, însă rezolvarea sa este cât se poate de simplă: joncţionarea celor două relaţii - vezi figura 2.20.

```
R <- JONCŢIUNE (LOCALITATI, JUDEŢE; Jud)
```

JUDEŢE

Jud	Jude!	Rfitjiune
IS	Iasi	Moldova
VN	Vrancea	Moldova
NT	Neamţ	Moldova
SV	Suceava	Moldova
VS	Vaslui	Moldova
TM	Timiş	Banat

LOCAUTATI

CoïlPos.	Loc	**Jud**
6600	Iasi	IS
5725	Paşcani	IS
6500	Vaslui	VS
5300	Focşani	VN
6400	Birlad	VS
5800	Suceava	SV
5550	Roman	NT
1900	Timişoara	TM

CodPost	Loc	**Jud**	**Judeţ**	**Reqiune**
6600	Iasi	IS	Iasi	Moldova
5725	Paşcani	IS	Iasi	Moldova
6500	Vaslui	VS	Vaslui	Moldova
5300	Focşani	VN	Vrancea	Moldova
6400	Birlad	VS	Vaslui	Moldova
5800	Suceava	SV	Suceava	Moldova
5550	Roman	NT	Neamţ	Moldova
1900	Timişoara	TM	Timiş	Banat

Figura 2.20. Joncţiunea tabelelor LOCALITĂŢI şi JUDEŢE - exemplul 13

Una din întrebările „clasice" pentru verificarea modului în care a fost sau nu înţeleasă joncţiunea este: *Câte linii are tabela-rezultat al joncţiunii*? în cazul nostru (de obicei în BDR) răspunsul este: *câte linii are tabela-copil.* Răspunsul este corect numai atunci când se respectă integritatea referenţială, altfel spus, numai atunci când toate valorile cheii străine se regăsesc în tabela-părinte.

Ca de obicei, lucrurile sunt mai complicate în realitate, deoarece joncţiunea nu se instituie musai între o tabelă-părinte şi una copil. Chiar primele exemple, cele „teoretice" - figurile 2.17, 2.18, 2.19 — folosesc două relaţii, Rl şi R2, despre care nu se face nici o afirmaţie privind filiaţia.

Exemplul 14

Care sunt localităţile din Banat?

[9] Soluţia 1

După calapodul exemplului anterior:

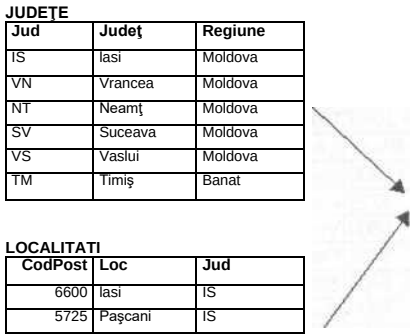
```
Rl JONCŢIUNE (LOCALITATI, JUDEŢE; Jud)
R SELECŢIE (Rl; Regiune = "Banat")
```

® Soluţia 2

Mai întâi se aplică selecţia asupra tabelei JUDEŢE, iar tabela intermediară se joncţionează cu LOCALITĂŢI:

```
Rl <r- SELECŢIE (JUDEŢE; Regiune = "Banat")
R <- JONCŢIUNE (LOCALITATI, Rl; Jud)
```

Care dintre cele două variante este de preferat? Răspunsul este facilitat de reprezentarea grafică a paşilor parcurşi în fiecare dintre cele două soluţii — vezi figurile 2.21 şi 2.22.



JUDEŢE

Jud	Judeţ	Regiune
IS	Iasi	Moldova
VN	Vrancea	Moldova
NT	Neamţ	Moldova
SV	Suceava	Moldova
VS	Vaslui	Moldova
TM	Timiş	Banat

LOCALITATI

CodPost	Loc	Jud
6600	Iasi	IS
5725	Paşcani	IS
6500	Vaslui	VS
5300	Focşani	VN
6400	Birlad	VS
5800	Suceava	SV
5550	Roman	NT
1900	Timişoara	TM

R1 - JONCŢIUNE (LOCALITATI, JUDEŢE ; Jud)

CodPost	Loc	Jud	Judeţ	Regiune
6600	Iasi	IS	Iasi	Moldova
5725	Paşcani	IS	Iasi	Moldova
6500	Vaslui	VS	Vaslui	Moldova
5300	Focşani	VN	Vrancea	Moldova
6400	Birlad	VS	Vaslui	Moldova
5800	Suceava	SV	Suceava	Moldova
5550	Roman	NT	Neamţ	Moldova
1900	Timişoara	TM	Timiş	Banat

CodPost	Loc	Jud	Judeţ	Regiune
1900	Timişoara	TM	Timiş	Banat

Figura 2.21. Plan de execuţie - exemplul 14, soluţia 1

JUDEŢE

Jud	Judeţ	Regiune
IS	Iasi	Moldova
VN	Vrancea	Moldova
NT	Neamţ	Moldova
SV	Suceava	Moldova
VS	Vaslui	Moldova
TM	Timiş	Banat

LOCALITATI

CodPost	Loc	Jud
6600	Iasi	IS
5725	Paşcani	IS
6500	Vaslui	VS
5300	Focşani	VN
6400	Birlad	VS
5800	Suceava	SV
5550	Roman	NT
1900	Timişoara	TM

Jud	Judeţ	Regiune
TM	Timiş	Banat

"Banat")

1

CodPost	Loc |Jud	Judeţ	Regiune
1900	Timişoara |TM	Timiş	Banat

Figura 2.22. Plan de execuţie - exemplul 14, soluţia 2

A doua variantă pare mai bună decât prima, deoarece joncţiunea operează asupra a două relaţii mai reduse ca dimensiuni. Diferenţa este cu atât mai vizibilă atunci când relaţia JUDEŢE conţine toate judeţele ţării, iar LOCALITĂŢI are câteva sute de înregistrări. Iar dacă ne gândim că înaintea oricărei joncţiuni se calculează produsul cartezian, apare drept firească ideea amânării joncţiunii, astfel încât aceasta să opereze asupra unor tabele cu un număr cât mai mic de linii şi coloane.

Coborând cu picioarele pe pământ, trebuie spus că discuţia noastră are un caracter de principiu, deoarece algebra relaţională este totuşi un limbaj... curat teoretic.

Exemplul 15

In ce zile s-a vândut produsul cu denumirea „Produs 1 ''?

Elementul de noutate îl reprezintă interesul pentru o informaţie ce provine dintr-o relaţie (atributul ataFact din FACTURI) pe baza unei condiţii aplicate altei relaţii (atributulDenPr din PRODUSE), iar cele două relaţii, FACTURI şi PRODUSE, nu sunt în raportul părinte-copil. în aceste cazuri, este necesară atragerea altor relaţii, până se completează „lanţul". Interogarea ce rezolva problema ridicată în acest exemplu necesită şi tabela LINIIFACT.

- Soluţie 1 - neoptimizată

```
Rl <- JONCŢIUNE (PRODUSE, LINIIFACT; CodPr)
R2   <- JONCŢIUNE   (Rl, FACTURI;        NrFact)
R3 <- SELECŢIE (R2; DenPr = "Produs 1")
R PROIECŢIE (R3; DataFact)
```

- Soluţie 2 - optimizată la sânge

```
Rl 4- SELECŢIE (PRODUSE; DenPr = "Produs 1")
R2   <- PROIECŢIE    (Rl; CodPr)
R3   <- JONCŢIUNE    (R2, LINIIFACT; CodPr)
R4   <- PROIECŢIE    (R3; NrFact)
R5      JONCŢIUNE    (R4, FACTURI;        NrFact)
R <- PROIECŢIE (R4; DataFact)
```

în cea de-a doua soluţie, fideli principiului, joncţiunii celei mai economicoase", am eliminat nu numai tuplurile, dar şi atributele de prisos înaintea calculării relaţiei intermediare.

Exemplul 16

în ce judeţe s-a vândut produsul cu denumirea „ Produs 1 '' in perioada 3-5 august 2000?

Relaţia-rezultat trebuie să conţină valori ale atributului Judeţ din tabela JUDEŢE. Predicatul de selecţie se aplică însă în alte două tabele: PRODUSE, în care DenPr =' „Produs 1", şi FACTURI, ale cărei tupluri trebuie să verifice condiţia: DataFact >= 03/08/2000 AND DataFact <= 03/08/2000.

```
Rl <- SELECŢIE (PRODUSE; DenPr = "Produs 1")
R2 <~ JONCŢIUNE (Rl, LINIIFACT; CodPr)
R3 PROIECŢIE (R2; NrFact)
R4 4- SELECŢIE(FACTURI; DataFact >= 03/08/2000 AND DataFact <=
   03/08/2000)
R5 <— JONCŢIUNE (R3, R4; NrFact)
R6 <- JONCŢIUNE (R5, CLIENŢI; CodCl)
R? <- PROIECŢIE (R6; CodPost)
R8 <- JONCŢIUNE (R7, LOCALITATI; CodPost)
R9 <- PROIECŢIE (R8; Jud)
R10 <- JONCŢIUNE (R9, JUDEŢE; Jud)
R <- PROIECŢIE (R10; Judeţ)
```

Exemplul 17

în ce zile s-au vândut şi produsul cu denumirea „ Produs 1 ", şi cel cu denumirea „ Produs 2 ? Rezultatul conţine atributul DataFact. Predicatul de selecţie se aplică tabelei

PRODUSE.

- Soluţie 1 "

```
Rl SELECŢIE (PRODUSE; DenPr = "Produs 1")
R2  *- JONCŢIUNE  (Rl, LINIIFACT; CodPr)
R3    JONCŢIUNE  (R2, FACTURI; NrFact)
R4  <- PROIECŢIE  (R3; DataFact)
R5 <- SELECŢIE (PRODUSE; DenPr = "Produs 2")
R6  <- JONCŢIUNE  (R5, LINIIFACT; CodPr)
R7  <- JONCŢIUNE  (R6, FACTURI; NrFact)
R8    PROIECŢIE  (R7; DataFact)
R <— R4 n R8
```

- Soluţie 2

```
Rl SELECŢIE (PRODUSE; DenPr = "Produs 1")
R2  4- JONCŢIUNE  (Rl, LINIIFACT; CodPr)
R3    JONCŢIUNE  (R2, FACTURI; NrFact)
R4 SELECŢIE (PRODUSE; DenPr = "Produs 2")
R5    JONCŢIUNE  (R5, LINIIFACT; CodPr)
R6  4- JONCŢIUNE  (R6, FACTURI; NrFact)
R7  <- JONCŢIUNE  (R3, R6; DataFact)
R <- PROIECŢIE (R7; DataFact)
```

Prima variantă este una „cuminte". Se intersectează relaţia zilelor în care s-a vândut primul produs (R4) cu relaţia zilelor în care s-a facturat produsul 2 (R8).

A doua e ceva mai insolită. Pur şi simplu, în loc de intersecţie folosim joncţiunea. Cum, spre deosebire de intersecţie, relaţiile joncţionate nu trebuie să fie unicompatibile, putem să operăm direct joncţiunea între R3 şi R6. .....

Reţinem ideea: putem să simulăm intersecţia a două relaţii prin joncţiune. Pentru a ti mai convingător, rogu-vă să examinaţi figura 2.23, în care, pornind de la relaţiile Rl şi R2, se obţine relaţia-intersecţie R direct prin operatorul intersecţie (stânga figurii) şi mai pe ocolite, folosind joncţiunea (dreapta).

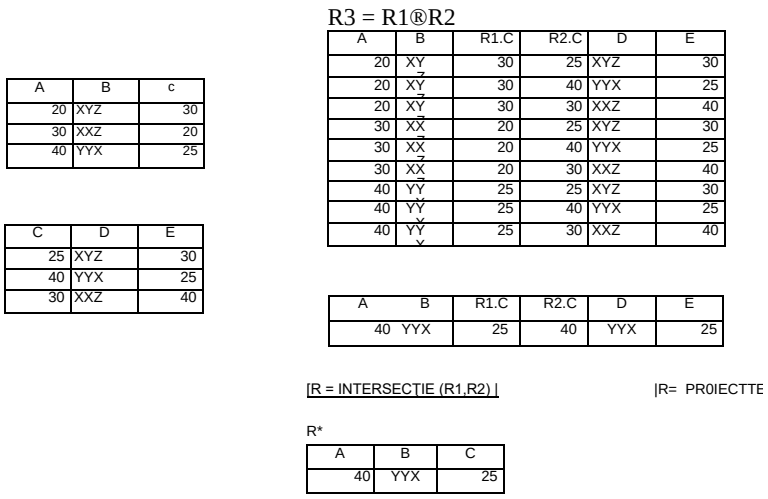

R3 = R1®R2
A B c
20 XYZ 30
30 XXZ 20
40 YYX 25
C D E
25 XYZ 30
40 YYX 25
30 XXZ 40
A B R1.C R2.C D E
20 XYZ 30 25 XYZ 30
20 XYZ 30 40 YYX 25
20 XYZ 30 30 XXZ 40
30 XXZ 20 25 XYZ 30
30 XXZ 20 40 YYX 25
30 XXZ 20 30 XXZ 40
40 YYX 25 25 XYZ 30
40 YYX 25 40 YYX 25
40 YYX 25 30 XXZ 40
A B R1.C R2.C D E
40 YYX 25 40 YYX 25
[R = INTERSECŢIE (R1,R2) |
|R= PR0IECTTE
R*
A B C
40 YYX 25

Figura 2.23. Intersecţia prin joncţiune

Exemplul 18

Ce clienţi au cumpărat,, Produs 2 " şi „ Produs 3 ", dar nu au cumpărat „ Produs 5 "?

Bună întrebare! Lucrurile nu sunt însă atât de complicate precum par, deoarece folosim elemente din exemplul anterior (intersecţia) plus operatorul... diferenţă.

```
R1  <- SELECŢIE  (PRODUSE; DenPr = "Produs 2"
R2  <- JONCŢIUNE  (R1  LINIIFACT; CodPr)
R3  <- JONCŢIUNE  (R2, FACTURI; NrFact)
R4  <— JONCŢIUNE  (R3  CLIENŢI; CodCl)
R5  <- PROIECŢIE  (R4  DenCl)
R6  <— SELECŢIE  (PRODUSE; DenPr = "Produs 3"
R7  <- JONCŢIUNE  (R6, LINIIFACT; CodPr)
R8  <— JONCŢIUNE  (R7, FACTURI; NrFact)
R9  <- JONCŢIUNE  (R8, CLIENŢI; CodCl)
R10 <- PROIECŢIE (R9; DenCl)
```

```
Rll <— SELECŢIE (PRODUSE; DenPr = "Produs 5")
R12  <- JONCŢIUNE (Rll, LINIIFACT; CodPr)
R13  4- JONCŢIUNE (R12, FACTURI;    NrFact)
R14  <- JONCŢIUNE (R13, CLIENŢI;     CodCl)
R15  4- PROIECŢIE (Rl4; DenCl)
R <- R5 n R10 - R15
```

Exemplul 19

Ce facturi au fost emise în aceeaşi zi cu factura 1120?

Spre deosebire de interogările de până acum, condiţia de selecţie este una indirectă. în prealabil, trebuie determinată ziua în care a fost întocmită factura cu numărul 1120. Apoi trebuie extrase, din relaţia FACTURI, liniile pentru care DataFact are valoarea zilei facturii-reper.

```
Rl 4- SELECŢIE (FACTURI; NrFact = 1120)
R2 4- PROIECŢIE (Rl; DataFact)
R3 <- JONCŢIUNE (FACTURI, R2; DataFact)
R 4- PROIECŢIE (R3; NrFact)
```

Alte două tipuri de joncţiune: joncţiunea externă şi semijoncţiunea

Cele trei tipuri de joncţiune prezentate (theta, echi, naturală) prezintă două extensii, *joncţiunea externă* şi *semijoncţiunea,* dintre care prima are o importanţă deosebită, mai ales prin transpunerea sa în SQL şi redactarea unor interogări elegante pentru probleme ceva mai complexe.

Joncţiunea externă

Ideea de bază a joncţiunii externe este de a include în rezultat şi tupluri din una dintre relaţii, sau din ambeie relaţii, care prezintă valori ale atributului de legătură ce nu se regăsesc în cealaltă relaţie.

Dacă precedentele tipuri de joncţiune sunt comutative, în cazul joncţiunii externe trebuie specificat din care relaţie se extrag liniile fără corespondent în cealaltă relaţie. De aceea, există *joncţiune externă la stânga* şi *joncţiune externă la dreapta.* La acestea se adaugă *joncţiunea externă totală* (denumită şi *plină* sau *deplina),* care reprezintă reuniunea celor două.

Apelând la aceleaşi tabele „abstracte", Rl şi R2, diferenţa dintre joncţiunea internă (echi-joncţiunea) şi cele trei tipuri de joncţiune externă apare cu mai multă claritate în figura 2.24.

Exemplul 20

Care sunt localităţile în care nu avem nici un client?

• Soluţia 1 se bazează pe diferenţa dintre tabela tuturor localităţilor (din tabela LOCALITĂŢI) şi tabela localităţilor în care există clienţi (tabela CLIENŢI).

```
•Rl 4- PROIECŢIE (LOCALITATI; CodPost)
R2 4- PROIECŢIE (CLIENŢI; CodPost)
R3 4- Rl - R2
R 4- JONCŢIUNE (R3, LOCALITATI; CodPost)
```

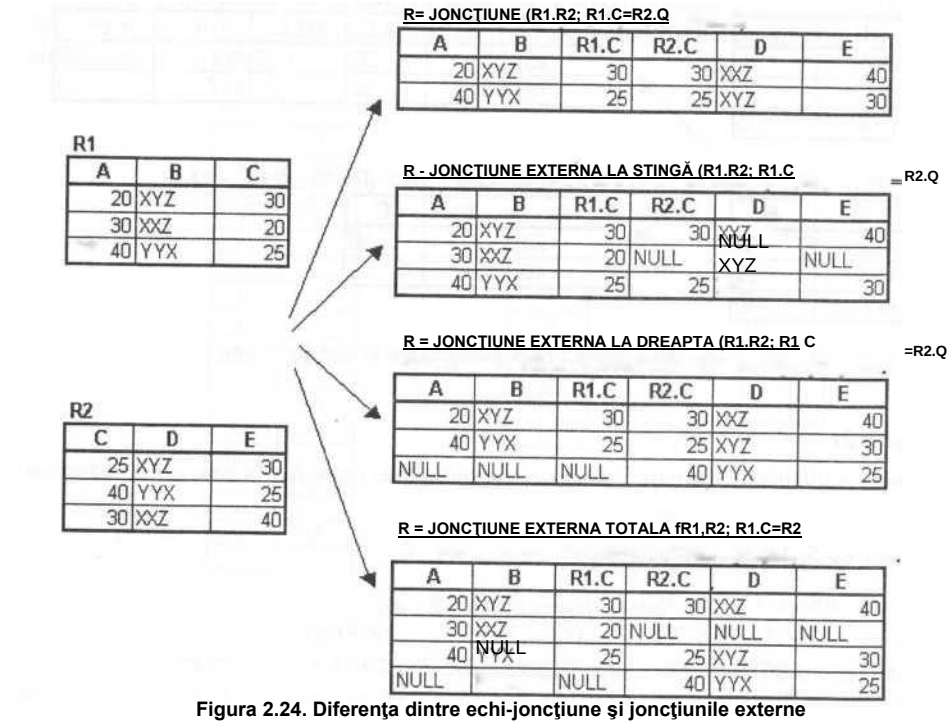

R1
A B C
20 XYZ 30
30 XXZ 20
40 YYX 25
R2
C D E
25 XYZ 30
40 YYX 25
30 XXZ 40
R= JONCŢIUNE (R1.R2; R1.C=R2.Q
A B R1.C R2.C D E
20 XYZ 30 30 XXZ 40
40 YYX 25 25 XYZ 30
R - JONCTIUNE EXTERNA LA STINGĂ (R1.R2; R1.C =R2.Q
A B R1.C R2.C D E
20 XYZ 30 30 XYZ 40
30 XXZ 20 NULL NULL NULL
40 YYX 25 25 30
R = JONCŢIUNE EXTERNA LA DREAPTA (R1.R2; R1 C =R2.Q
A B R1.C R2.C D E
20 XYZ 30 30 XXZ 40
40 YYX 25 25 XYZ 30
NULL NULL NULL 40 YYX 25
R = JONCŢIUNE EXTERNA TOTALA fR1,R2; R1.C=R2
A B R1.C R2.C D E
20 XYZ 30 30 XXZ 40
30 XXZ 20 NULL NULL NULL
40 YYX 25 25 XYZ 30
NULL NULL 40 YYX 25

Figura 2.24. Diferenţa dintre echi-joncţiune şi joncţiunile externe

- Soluţia 2 utilizează proaspăta joncţiune externă.

```
Rl <- JONCŢIUNE EXTERNĂ LA STÎNGA (LOCALITATI, CLIENŢI;
    LOCALITATI.CodPost = CLIENŢI.CodPost)
 R <- SELECŢIE (Rl; CodCl IS NULL)
```

Pentru a identifica localităţile fară clienţi, s-au extras, din joncţiunea externă la stânga a relaţiilor LOCALITAŢI şi CLIENŢI, numai liniile în care unul din atributele preluate din CLIENŢI (noi ne-am oprit asupra primului, `CodCl`) are valoarea NULL.

Semijoncţiunea

Scmijoncţiunea este unul dintre cele mai „marginalizate" tipuri de joncţiune. Implementarea sa în SGBD-urile comerciale este extrem de rară. A fost introdusă din dorinţa de a optimiza procesul de interogare (consultare). Calculul semijoncţiunii a două tabele presupune *selectarea numai a liniilor din prima tabelă care apar în joncţiune cu linii din a doua tabelă* - vezi figura 2.25.

A	B	C
20	XYZ	30
30	XXZ	20
40	YYX	25

R2

C	D	E
25	XYZ	30
40	YYX	25
30	XXZ	40

R= JONCŢIUNE R1.R2; R1.C =R2.Q

A	B	R1.C	R2.C	D	E
20	XYZ	30	30	XXZ	40
40	YYX	25	25	XYZ	30

R= SEMIJONCŢIUNE (R1 ,R2; R1.C=R2.Q

A	B	C
20	XYZ	30
40	YYX	25

Figura 2.25. Diferenţa dintre echi-joncţiune şi semijoncţiune

Exemplul 21

Care sunt localităţile (codul poştal, denumirea şi indicativul judeţului) în care există măcar un client?

- Soluţia 1 - bazată pe echi-joncţiune

```
Rl <- PROIECŢIE (CLIENŢI; CodPost)
R2 <- JONCŢIUNE (Rl, LOCALITATI; CodPost)
R <- PROIECŢIE (R2; CodPost, Loc, Jud)
```

- Soluţia 2 - bazată pe semijoncţiune

```
R 4- SEMIJONCŢIUNE (LOCALITATI, CLIENŢI; CodPost)
```

Ca şi cum nu ar fi fost de ajuns, în „literatura" relaţională mai există un tip de joncţiune - *autojoncţiunea* (închiderea tranzitivă), utilă în procesul de elaborare a bazelor de date relaţionale, la normalizarea relaţiilor[34], pe care o lăsăm însă în plata teoriei.

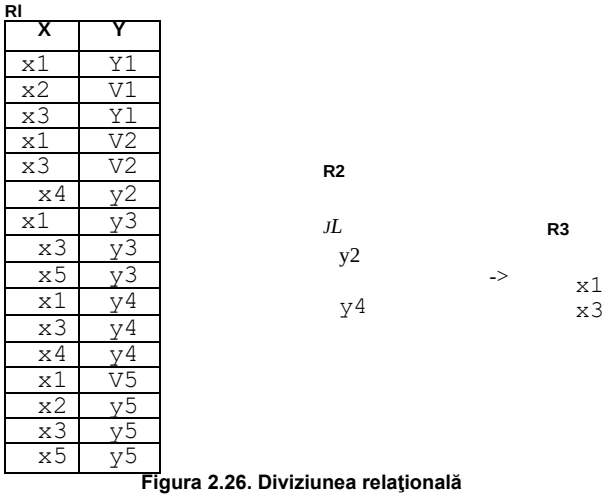
Diviziunea

Este cel mai complex şi mai greu de explicat dintre operatorii prezentaţi în acest capitol. Codd l-a imaginat ca operator invers al produsului cartezian. Pentru a-1 defini, se porneşte de la două relaţii R1(X, Y) şi R2(Y); prima are, care vasăzică, două atribute sau grupe de atribute, notate X şi Y, în timp ce a doua numai atributul sau grupul de atribute notat cu Y (definit pe acelaşi domeniu ca şi în relaţia Rl).

O primă restricţie: relaţia R2(Y), fiind numitorul diviziunii, nu trebuie să fie vidă. Diviziunea relaţională Rl -s- R2 are ca rezultat o relaţie definită ca ansamblul sub-tuplurilor R1(X) pentru care produsul (lor) cartezian cu R2(Y) este un subansamblu al R1(X, `Y).` Rezultatul expresiei Rl `-s-` R2 reprezintă *câtul* diviziunii, fiind o relaţie ce poate fi notată R3(X). într-o altă formulare, xi e R3 dacă şi numai dacă `V y i e Y e R 2 3 (xi, yi) e R l.`

34. [Lungu ş.a. 95], pp. 114-115.

Pentru simplificarea prezentării, în continuare am considerat `X` şi `Y` două atribute, deşi, după cum reiese din preambul, acestea pot fi grupe (ansambluri) de atribute. Să examinăm elementele din figura 2.26.



Rl

X	Y
x1	Y1
x2	V1
x3	Y1
x1	V2
x3	V2
x4	y2
x1	y3
x3	y3
x5	y3
x1	y4
x3	y4
x4	y4
x1	V5
x2	y5
x3	y5
x5	y5

Figura 2.26. Diviziunea relaţională

Determinarea relaţiei R3 <— Rl R2 este sinonimă cu rezolvarea problemei: *care dintre* `xl, x2, x3, x4` *şi* `x5` *apar în* ***Rl,*** *în tupluri împreună cu* ***toate*** *valorile lui* `Y` *din* ***R2,*** *respectiv* `yl, y2, y3, y4` *şi* `y5` ?

Se parcurg pe rând valorile `xi` ale atributului `X` din relaţia Rl:

- `xl` apare cu `yl` (în tuplul 1), cu `y2` (în tuplul 4), cu `y3` (în tuplul **7),** cu `y4` (în tuplul 10) şi cu `y5` (în tuplul 13). Deci `xl` îndeplineşte condiţia şi va fi inclus în relaţia R3;
- `x2` apare cu `yl` (în tuplul 2) dar nu apare cu `y2` - *nu* va face parte din R3;
- `x3` apare cu `yl` (în tuplul 3), cu `y2` (în tuplul 5), cu`y3` (în tuplul 8), cu `y4` (în tu şi cu y_5 (în tuplul 15) - îndeplineşte condiţia şi va fi tuplu în R3;
- `x4` nu apare cu `yl` - *nu* \a face parte din R3.
- `x5` nu apare cu `yl` - *nu* va face parte din R3.

în urma raţionamentului de mai sus, tabela R3 va fi alcătuită din două tupluri. Deşi pare un operator ceva mai metafizic, diviziunea relaţională este deosebit de utilă pentru formularea consultărilor în care apare clauza „V" („oricare ar fi" sau „pentru toate").

Exemplul 22

Care sunt clienţii pentru care există cel puţin câte o factură emisă în fiecare zi?

într-o altă formulare, ne interesează *clienţii* care au cumpărat „câte ceva" în *toate zilele* în care s-au efectuat vânzări. Prin urmare, câtul va fi o tabelă cu singur atribut, DenCl (denumirea clientului), iar divizorul va fi o relaţie alcătuită numai din atributul DataFact (conţine toate zilele în care s-au efectuat vânzări). Dacă am merge pe calapodul prezentat, am putea nota R3(DenCl), R2(DataFact). Cunoscând structura câtului şi a divizorului, putem determina structura tabelei-deîmpărţit: Rl(DenCl, DataFact). Relaţia Rl va conţine denumirile clienţilor şi zilele în care există măcar o factură pentru clientul respectiv.

Soluţia poate fi redactată în următorii paşi:

- construire relaţie-deîmpărţit:

```
Rll <- JONCŢIUNE (FACTURI, CLIENŢI; CodCl)
Rl 4- PROIECŢIE (Rl; DenCl, DataFact)
```

- construire relaţie-numitor:

```
R2 4- PROIECŢIE (FACTURI; DataFact)
```

- în fine, apoteoza:

```
R3 <- Rl * R2
```

Schema şi conţinutul relaţiilor implicate în această soluţie sunt prezentate în figura 2.27.

rIll NrFact		CodCl	DenCl	CodFiscal	Adresa	CodPost
1111	01/08/2000	1001	Client 1 SRL	R1001	Tranziţiei, 13 bis	6600
1112	01/08/2000	1005	Client 5 SRL	R1005	NULL	1900
1113	01/08/2000	1002	Client 2 SA	R1002	NULL	6600
1114	01/08/2000	1006	Client 6 SA	R1006	Pacientei, 33	5550
1115	02/08/2000	1001	Client 1 SRL	R1001	Tranziţiei, 13 bis	6600
1116	02/08/2000	1007	Client 7 SRL	R1007	Victoria Capitalismului, 2	1900
1117	03/08/2000	1001		R1001	Tranziţiei, 13 bis	6600
1118	04/08/2000	1001	Client 1 SRL	R1001	Tranziţiei, 13 bis	6600
1119	07/08/2000	1003	Client 3 SRL	R1003	Prosperităţii, 22	6500
1120	07/08/2000	1001	Client 1 SRL	R1001	Tranziţiei, 13 bis	6600
1121	07/08/2000	1004	Client 4	NULL	Sapientei, 56	5725

NrFact	DataFact	CodCl	Obs
1111	01/08/2000	1001	NULL
1112	01/08/2000	1005	Probleme cu transportul
1113	01/08/2000	1002	NULL
1114	01/08/2000	1006	NULL
1115	02/08/2000	1001	NULL
1116	02/08/2000	1007	Preţul propus iniţial a fost modificat
1117	03/08/2000	1001	NULL
1118	04/08/2000	1001	NULL
1119	07/08/2000	1003	NULL
1120	07/08/2000	1001	NULL
1121	07/08/2000	1004	NULL .
1122	07/08/2000	1005	NULL

DenCl	DataFact
Client 1 SRL	01/08/2000
Client 5 SRL	01/08/2000
Client 2 SA	01/08/2000
Client 6 SA	01/08/2000
Client 1 SRL	02/08/2000
Client 7 SRL	02/08/2000
Client 1 SRL	03/08/2000
Client 1 SRL	04/08/2000
Client 3 SRL	07/08/2000
Client 1 SRL	07/08/2000
Client 4	07/08/2000

R2
DataFact
01/08/2000
02/08/2000
03/08/2000
04/08/2000
07/08/2000

R3
DenCl
I Client 1 SRL |

Figura 2.27. Diviziunea relaţională - exemplul 22

Poate o să spuneţi că n-a meritat efortul, dar acest gen de soluţii se aplică la o gamă largă de probleme. Să discutăm câteva speţe.

Exemplul 23

în ce zile s-au vândut şi produsul cu denumirea „ Produs 1", şi cel cu cţenumirea „ Produs 2 ?

Vă gândiţi, probabil, că aţi mai văzut undeva acest enunţ. Este chiar textul problemei de la exemplul 17. La acel moment am formulat două soluţii: una „clasică', bazată pe intersecţie, şi una mai neconvenţională ce utilizează joncţiunea. Adăugăm la acestea o alta, încadrabilă probabil tot în categoria „neconvenţionale".

- Soluţie 3

```
Rll 4- SELECŢIE (PRODUSE; DenPr = "Produs 1" OR DenPr =
"Produs 2")
```

```
R12 <- JONCŢIUNE (Rll, LINIIFACT; CodPr)
R13 4- JONCŢIUNE (R12, FACTURI; NrFact)
Rl <— PROIECŢIE (R13; DataFact, CodPr)
R2 4- PROIECŢIE (Rll; CodPr)
R3 <— Rl -f- R2
```

Relaţia-divizor este alcătuită din două tupluri ce conţin codurile celor două produse. Deîmpărţitul este alcătuit din atributele `DataFact` şi `CodProd` şi conţine toate produsele vândute în fiecare din zilele de facturare.

Exemplul 24

Deşi reprezintă un regres vizibil faţă de nivelul interogărilor precedente, revenim discret la enunţul exemplului 19.

Ce facturi au fost emise în aceeaşi zi cu factura 1120?
Pentru această problemă se poate încropi şi o variantă de rezolvare bazată pe diviziunea relaţională.

```
Rl  4-PROIECŢIE  (FACTURI; NrFact,   DataFact)
R2' <- SELECŢIE (FACTURI; NrFact = 1120)
R2    PROIECŢIE  (R2'; DataFact)
R 4- DIVIZIUNE (Rl, R2)
```

Exemplul 25

Căror clienţi le-au fost vândute toate produsele firmei?

```
Rll 4- JONCŢIUNE (LINIIFACT, FACTURI; NrFact)
Rl    PROIECŢIE  (Rll; CodCl, CodPr)
R2  4-PROIECŢIE  (PRODUSE; CodPr)
R3  <-DIVIZIUNE  (Rl, R2)
R <- JONCŢIUNE (R3, CLIENŢI; CodCl)
```

Diviziunea relaţională nu este un operator fundamental; funcţionalitatea sa poate fi realizată prin combinarea operatorilor: produs cartezian, diferenţă şi proiecţie.

Se reiau în discuţie atât tabelele Rl şi R2 din figura 2.26, cât şi problema: *care dintre valorile* `xi` *•apar în tupluri, în Rl, cu toate valorile* y j *din R2?*

O soluţie a problemei poate fi constituită din următorii paşi:

- `Rll <- PROIECŢIE (Rl; X)`
- `R12 <r- Rll (S> R2.` Tabela R12 cuprinde toate tuplurile posibile `(xi,` yj)
- `R13 <— R12 -Rl.` Interesează ce tupluri din Rl2 lipsesc în Rl
- Valorile `xi` din R13 nu apar în Rl în combinaţie cu <u>toate</u> valorile yj din R2. Se elimină dublurile prin `Rl4 PROIECŢIE (R13; X)`
- Valorile `xi` din R14 sunt cele care nu prezintă tupluri obţinute prin combinaţii cu toate valorile y j. Deoarece interesează cele care <u>prezintă</u> combinaţiile respective, rezultatul se obţine prin diferenţa `R3 <— Rll - R14.`

Figura 2.28 aduce, cât de cât, un plus de claritate.

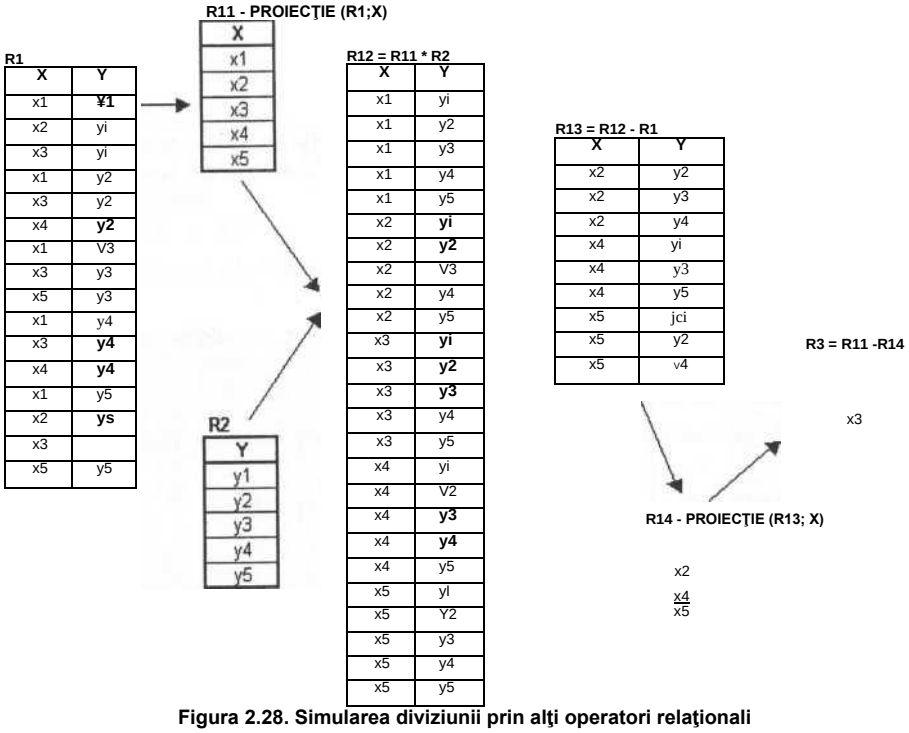


Figura 2.28. Simularea diviziunii prin alţi operatori relaţionali

Dintre operatorii prezentaţi pe parcursul ultimelor două paragrafe, selecţia, proiecţia, produsul cartezian, reuniunea şi diferenţa sunt operatori *fundamentali* sau *ireductibili,* în sensul că nici unul nu poate fi definit prin intermediul celorlalţi.

- Joncţiunea, intersecţia şi diviziunea sunt operatori *derivaţi*;
- Joncţiunea se exprimă cu ajutorul produsului cartezian urmat de selecţie;
- Intersecţia se exprimă cu ajutorul diferenţei, conform relaţiei

```
Rl n R2 = Rl - (Rl - R2) ;
```

- Diviziunea poate fi exprimată, aşa cum am văzut ceva mai sus, prin intermediul produsului cartezian, proiecţiei şi diferenţei.

2.4. Alte notaţii şi reprezentarea grafică ale interogărilor

încă de la începutul prezentării operatorilor algebrei relaţionale, precizam că notaţia pe care o vom folosi este una dintre cele mai comode (nu şi cea mai uzuală). Modelul relaţional este, în ciuda materialului de faţă, unul matematizat. Algebra relaţională este, aşa

cum îi spune şi numele, riguros formalizată. De aceea, şi pentru un „look" mai ştiinţific, majoritatea lucrărilor folosesc o notaţie matematică sau... aproximativ matematică.

Notaţia matematică

Există o multitudine de lucrări care utilizează această notaţie. în cazul de faţă, sursa principală o constituie traducerea franceză o unei lucrări mai vechi de Korth şi Silberschatz[35]. Notaţia prezentată foloseşte, pe lângă clasicele:

- u - reuniune,
- n - intersecţie,

® x - produs cartezian,

- -5- - diviziune

şi simboluri din alfabetul grecesc, după cum urmează:

» pentru selecţie: CT_P (R), unde P este un predicat aplicat asupra relaţiei R,

® pentru proiecţie: Ils (R), unde S este o listă de atribute din R,

® pentru joncţiune: Rl >< $_{MA>R1C}$ R2 - indică (theta)joncţiunea relaţiilor Rl şi R2 prin predicatul: Rl.A > R2.C.

Exemple

Soluţia de la exemplul 7:

```
R1 <- SELECŢIE (CLIENŢI; DenCl = "Client 2 SA")
R2 <- PROIECŢIE (Rl; Telefon)                    '
```

poate fi transcrisă în notaţia matematică după cum urmează:

$$n_T e, e_{fo}, (a_{Den} a = .c_{ll} e, asv(^{CLIENTI}))$$

După cum se observă, întreaga interogare se scrie într-o singură expresie.

- Soluţia de la exemplul 9:

```
R1 <-SELECŢIE (LINIIFACT; NrFact   = 1111)
R2   PROIECŢIE (Rl; CodPr)
R3 <-SELECŢIE (LINIIFACT; NrFact   = 1117)
R4 <-PROIECŢIE (R3; CodPr)
R5 4—R2 O R4 :
```

$$ELdp/a\ NrFact{=}l\ 111\ (LINIIFACT))\ n\ ricodPr\ (O_{NrFaa=l,\ 17}\ (LINIIFACT))$$

- Soluţia 1 de la exemplul 14:

```
R1   JONCŢIUNE (LOCALITATI, JUDEŢE; Jud)
R <- SELECŢIE (Rl; Regiune = "Banat")
```

$$a_{Regi\_.Ba,a,}.(LOCALITATI >< _{L0CALITATIM=,,Jud}\ JUDEŢE)$$

35. [Korth&Silberschatz88],

Este drept, atunci când trebuie reprezentată o joncţiune naturală, se poate recurge la o redactare simplificată:

rr (LOCALITATI ><1 JUDEŢE)
vJ Regiune="Banat" ˅

O altă observaţie ţine de faptul că optimizarea interogărilor nu prezintă aproape deloc importanţă.

- Soluţia de la exemplul 16

```
R1 <- SELECŢIE (PRODUSE; DenPr = "Produs 1")
R2 <r- JONCŢIUNE (R1, LINIIFACT; CodPr)
R3  <r- PROIECŢIE  (R2; NrFact)
R4       SELECŢIE(FACTURI; DataFact >=       03/08/2000 AND
DataFact <= 03/08/2000)
R5    JONCŢIUNE  (R3, R4; NrFact)
R6    JONCŢIUNE  (R5, CLIENŢI; CodCl)
R7    PROIECŢIE  (R6; CodPost)
R8  <- JONCŢIUNE  (R7, LOCALITATI;  CodPost)
R9  4- PROIECŢIE  (R8; Jud)
R10 <- JONCŢIUNE (R9, JUDEŢE; Jud)
R PROIECŢIE (R10; Judeţ):
      n ( „                                                    (PRODUSE r><
      1 1 Judeţ      DenPr="Produs 1"  a      DataFaci >=03/08/2000      aDataFact      <=05/08/2000      ˅      K ^
                                  LINIIFACT>o FACTURI>< CLIENŢI>< LOCALITATI[><| JUDEŢE))
```

Nu spun că acest stil de notare ar fi din cale-afară de dificil, dar eu, unul, consider că varianta „noastră" de redactare este mult mai lejeră.

Gramatica BNF

Notaţia următoare este preluată din ediţia a 4-a a lucrării lui C.J. Date şi, cu oarecare amendamente din partea lui Date, reprezintă gramatica *Backus Naur Form.* Avantajele acestei notaţii ţin, în primul rând, de utilizarea unor cuvinte din limba engleză, în locul abstractelor simboluri matematice. De asemenea, interogările se scriu liniar, fund astfel mai lizibile. Pentru a indica precedenţa operaţiilor se folosesc parantezele. Un avantaj major faţă de notaţia folosită pe parcursul acestui capitol ţine de faptul că pune în evidenţă modul în' care operaţiile sunt incluse unele în altele, fiind mai aproape şi de logica SQL (care foloseşte o singură frază SELECT în care, eventual, sunt incluse subconsultări).

Fără a intra în detalii privind gramatica BNF tratată *in extenso* în lucrarea autorului american, amintim că reprezentarea operatorilor se face astfel:

- *reuniune:* `R1 UNION R2`
- *intersecţie:* `R1 INTERSECT R2`
- *diferenţă'.* `R1 MINUS R2`
- *produs cartezian'.* `R1 TIMES R2`
- *selecţie,* denumită şi *restricţie:* `R WHERE P`, unde *P* este un predicat aplicat asupra relaţiei R
- *proiecţie:* `R [A, B, ... Z]`, unde `A, B, ...Z` sunt atribute ale `R`
- *theta-joncţiunea* se reprezintă diferit de *joncţiunea naturală,* ca un operator compus dintr-un produs cartezian urmat de o selecţie. Astfel, pentru (theta)joncţiunea relaţiilor R1 şi R2 prin

22. [Date86], p. 89.

predicatul: `Rl. A > R2 . C,` se foloseşte notaţia:
`Rl TIMES R2) WHERE Rl.A > R2.C`

- pentru *joncţiunea naturală* lucrurile sunt mai simple; astfel, joncţiunea naturală dintre Rl şi R2 prin atributul C (atributul comun) se simbolizează pur şi simplu: `Rl JOIN R2.`
- diviziune: `Rl DIVIDED BY R2`

Exemple

- Soluţia de la exemplul 7:

```
(CLIENŢI WHERE DenCl = "Client 2 SA") [Telefon]
```

- Soluţia de la exemplul 9:

```
( ( LINIIFACT WHERE NrFact = 1111 ) [CodPr] ) INTERSECT (
(LINIIFACT WHERE NrFact = 1111 ) [CodPr] )
```

- Soluţia 1 de la exemplul 14:

```
( ( JUDEŢE JOIN LOCALITATI ) WHERE Regiune = "Banat" )
```

- Soluţia de la exemplul 16

```
( ( ( ( ( ( ( ( PRODUSE WHERE DenPr = "Produs 1" ) JOIN
LINIIFACT ) [NrFact] ) JOIN FACTURI ) WHERE DataFact >=
03/08/2000 AND DataFact <= 03/08/2000 ) JOIN CLIENŢI ) JOIN
LOCALITATI ) JOIN JUDEŢE ) [Judeţ]) '
```

Probabil cel mai mare dezavantaj al ecestei notaţii ţine de faptul că s-ar putea să greşim la număratul parantezelor...

Notaţii diverse

Literatura de specialitate a propus diverse notaţii, mai mult sau mai puţin apropiate de cele prezentate până acum. Dacă e să ne referim numai la cărţile publicate la noi[17] (bazate, ca şi prezenta, pe lucrări de circulaţie internaţională), putem aminti:
pentru reuniune:

- `Rl U R2`
- `OR (Rl, R2)`
- `APPEND (Rl, R2)`
- `UNION (Rl, R2)`
- `REUNIUNE (Rl,R2)`

pentru intersecţie:

- `Rl n R2`
- `AND (Rl, R2)`
- `INTERSECT (Rl, R2)`
- `INTERSECŢIE (R1,R2)`

pentru diferenţă:

- `Rl - R2`
- `REMOVE (Rl, R2)`
- `MINUS (Rl, R2)`
- `DIFERENŢĂ (R1,R2)`

17 [Lungu ş.a. 95], [Popescu96], [Dollinger98], [Florescu ş.a. 99] (în treacăt fie spus, exemplul de la diviziunea relaţională, p. 143, e cel puţin tulburător), [Grama&FilipOO],

pentru produs cartezian:

- `Rl x R2`
- `PRODUCT (Rl, R2)`
- `TIMES (Rl, R2)`
- `PROD (Rl,R2)`

pentru selecţie:

- `acondiţie (R)`
- `R [condiţie]`
- `RESTRICT (R, condiţie)`
- `SEL (R, condiţie)`

pentru proiecţie:

- riA, `B, C, ...Z (R)`
- `R [A, B, Z]`
- `PROJECT (R, A, B, ....)`
- `PROI (R, A, B, ..., Z)`

pentru joncţiune:

- Rl >_{conditic} R2
- `JOIN (Rl, R2, condiţie)`

pentru diviziune:

- `Rl * R2`
- `DIVISION (Rl, R2)`

Reprezentarea grafică a interogărilor

în multe lucrări de specialitate sunt utilizate o serie de simboluri grafice asociate operatorilor relaţionali. Dintre acestea, cele mai importante sunt prezentate în figura 2.29.

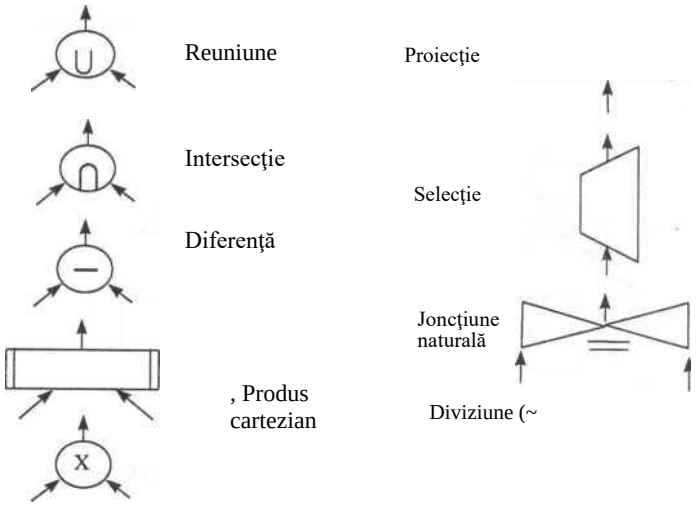


Figura 2.29. Simboluri pentru reprezentarea grafică a operatorilor algebrei relaţionale

Aceste simboluri permit reprezentarea, în condiţii de lizibilitate sensibil îmbunătăţită, a logicii de derulare a unei interogări formulate prin operatorii relaţionali. Pentru ilustrarea dispunerii simbolurilor în concordanţă cu logica soluţiei, reluăm cele patru exemple folosite în acest paragraf.

Exemple

- Soluţia de la exemplul 7 - figura 2.30:

R

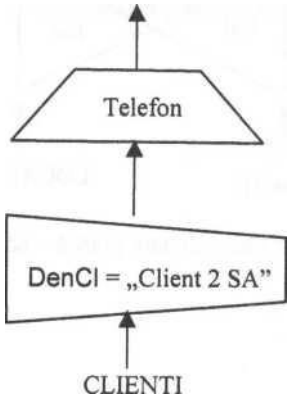


Figura 2.30. Reprezentare grafică - soluţie ex. 7

- Soluţia de la exemplul 9 - figura 2.31:

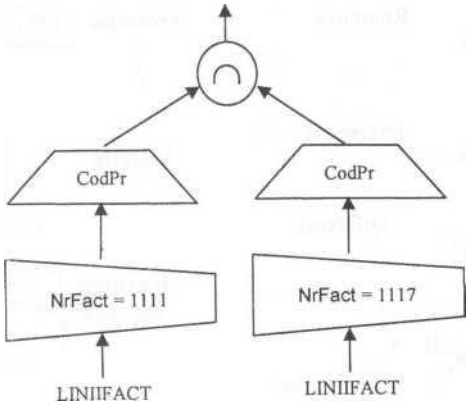


Figura 2.31. Reprezentare grafică - soluţie ex. 9

- Soluţia 1 de la exemplul 14 - figura 2.32:

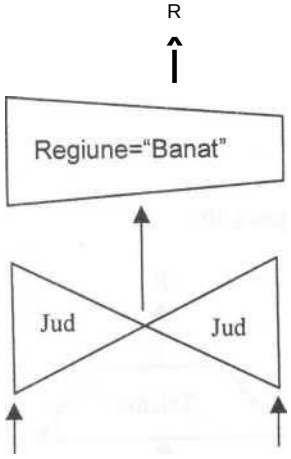


Figura 2.32. Reprezentare grafică - soluţie 1 ex. 14

• Soluţia de la exemplul 16 - figura 2.33:

R

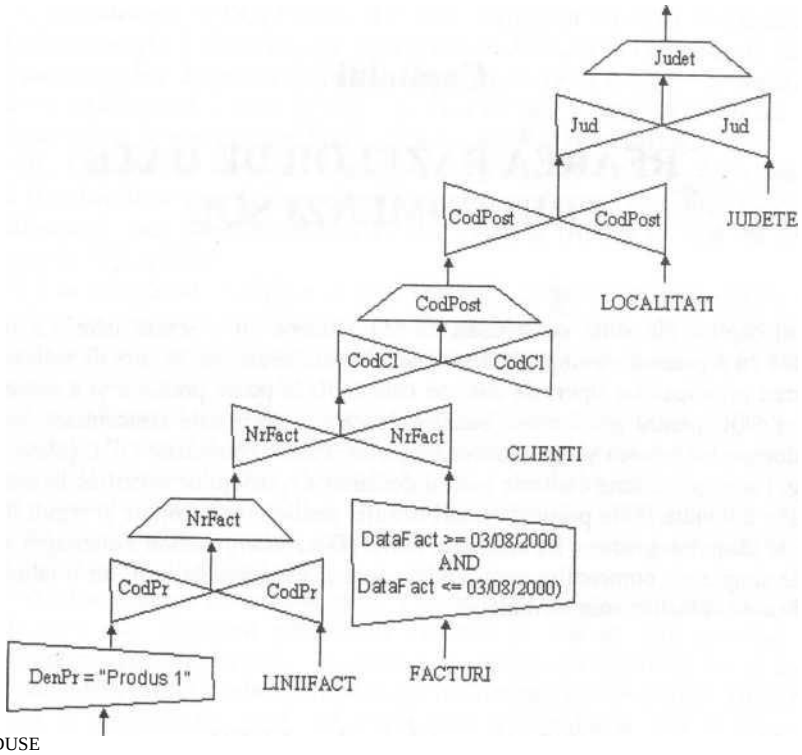


Figura 2.33. Reprezentare grafică - soluţie ex. 16

Capitolul 3

CREAREA BAZELOR DE DATE PRIN COMENZI SQL

Primul capitol din suita celor dedicate SQL începe cu o scurtă istorie a limbajului, completată cu o prezentare sintetică a standardelor elaborate sau în curs de elaborare. După enumerarea principalelor tipuri de date pe care SQL le poate prelucra şi a categoriilor de operaţiuni SQL pentru gestionarea bazelor de date, atenţia este concentrată pe tipologia comenzilor pentru crearea şi actualizarea tabelelor. Pentru comenzile DDL *(Data Defmition Language)* sunt prezentate clauzele pentru declararea restricţiilor specifice bazelor de date relaţionale: unicitate/cheie primară, valori nenule, restricţii referenţiale şi reguli de validare la nivel de câmp/înregistrare. La categoria DML *(Data Manipulation Language)* se prezintă formatele simple ale comenzilor pentru adăugarea şi ştergerea de linii într-o tabelă, precum şi modificarea valorilor unor atribute.

3.1. Scurt istoric al SQL

Utilizând mult îndrăgita şi încercata „limbă de lemn", pentru care dispunem de un *know-how* greu de egalat, se poate afirma că SQL a devenit, de o bună bucată de vreme, stindardul limbajelor din a IV-a generaţie. Nu există astăzi lucrări sau publicaţii în domeniul SGBD-urilor care să nu prezinte

mecanismul de lucru al acestui limbaj sau ultimele produse/tendinţe din lumea SQL. Impactul său este profund. Devenit un soi de esperando al limbajelor pentru baze de date, SQL are toate şansele de a nu cădea în desuetudinea esperando-ului propriu-zis, aceasta deoarece a fost altoit pe toate tipurile de SGBD-uri, de la cele dedicate microcalculatoarelor (PC), la cele care operează în medii distribuite/eterogene/client-server.

SQL este, pe de o parte, unul dintre „responsabilii" noii apropieri a non-informati- cianului de datele sale, iar pe de altă parte, pentru profesionişti, reprezintă nucleul dezvoltării aplicaţiilor ce utilizează bazele de date.

Dacă, încă din 1970, E.F. Codd sugera „adoptarea unui model relaţional pentru organizarea datelor [...] care să permită punerea la punct a unui sub-limbaj universal pentru gestiunea acestora, sub-limbaj care să fie, în fapt, o formă aplicată de calcul asupra predicatelor"[38], momentul decisiv în naşterea SQL îl constituie lansarea proiectului *System/R* de către firma IBM, eveniment ce a avut loc în 1974.

Tot în 1974, Chamberlin şi Boyce au publicat un articol[18] în care este prezentat un limbaj structurat de interogare, denumit *SEQUEL {Structured English as QUery Language).* In 1975, Chamberlin, Boyce, King şi Hammer redactează o lucrare dedicată sub-limbajului *SQUAREr* , asemănător SEQUEL-ului, dar care utilizează expresii matematice şi nu cuvinte din limba engleză. Autorii celor două studii au demonstrat că limbajele *SEQUEL* şi *SQUARE* sunt complete din punct de vedere relaţional. în 1976, o echipă de autori condusă de Chamberlin elaborează o nouă lucrare[19] în care se face referire la *SEQUEL 2,* acesta fiind declarat limbaj de interogare al SGBD-ului *System/R* al firmei IBM.

în 1980, Chamberlin schimbă denumirea *SEQUEL* în *SQL* - Structured Query Language (Limbaj Structurat de Interogare)[20]*, dar şi astăzi mulţi specialişti pronunţă SQL ca pe predecesorul său. Don Chamberlin lucrează încă la IBM şi publică, în continuare, lucrări legate de SQL şi DB2.

Anii '80 au înregistrat apariţia a o serie întreagă de lucrări dedicate SQL, care l-au perfecţionat şi consacrat ca pe cel mai răspândit limbaj de interogare a BDR, prezent în numeroase „dialecte" specifice tuturor SGBDR-urilor actuale, de la *DB2* la *Microsoft SQL Server,* de la *Oracle* Ia *FoxPro* şi *Access.*

încercând să răspundă solicitărilor privind standardizarea unui limbaj de lucru cu bazele de date, Institutul Naţional American pentru Standarde (ANSI) a încredinţat această sarcină comitetului X3H2 în anul 1982. Comitetul pentru baze de date X3H2 (unul dintre numeroasele consilii tehnice ale X3) a fost şi este format din experţi independenţi şi din aproape toate firmele importante din domeniul bazelor de date. Figura 3.1 ilustrează poziţia X3H2 în organismele de standardizare[21].

Un standard nou începe cu o schiţă, *proiect de lucru (working draft)* întocmit de comitet (XjH2) şi care este ameliorat pană cand membrii convin că este aproape finalizat, în această fază proiectul de lucru devine *proiect al comitetului (committee draft),* care este transmis comunităţii profesionale şi firmelor pentru comentarii şi propuneri. După preluarea observaţiilor şi propunerilor, proiectul comitetului este publicat sub titulatura *proiect internaţional de standard* şi, în final, *standard internaţional.*

Din 1982, rapid (în circa doi ani), au fost elaborate câteva versiuni ale standardului care însă

18 [Chamberlin&Boyce74],

19 [Chamberlin s.a. 76].

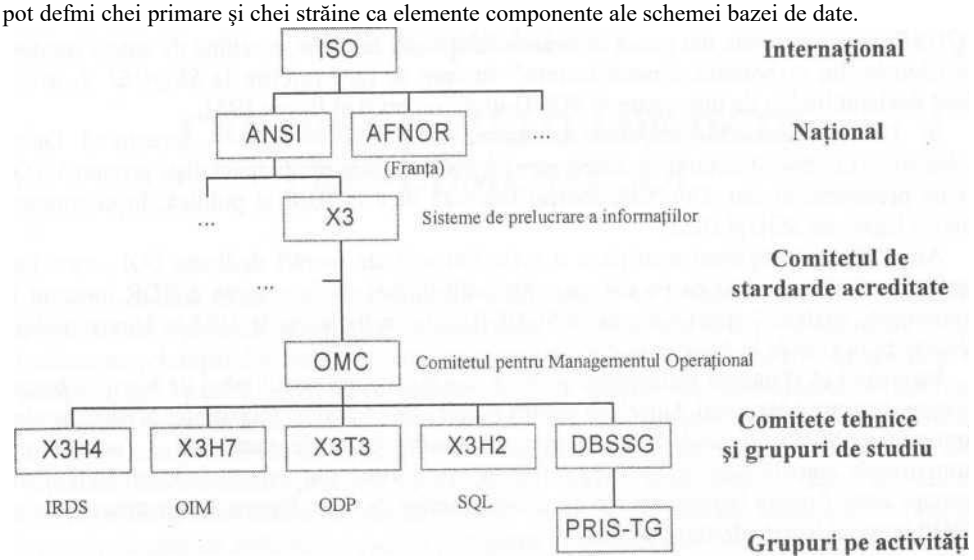
20 Se pare ca a fost o problema şi de Copyright asupra sintagmei SEQUEL - vezi şi [Hemandez& ViescaOO], p. 52.

21 Preluare din [Fortier99], p. 4.

devenise incompatibil cu dialectele SQL majore la acel moment, astfel încât a fost necesară reluarea lucrului pentru asigurarea unui minim numitor comun.

ANSI publică în 1986 standardul *SQL ANSIX3.135-1986,* standard ce utilizează, pentru reprezentarea sintaxei, forma Backus-Naur. Este un standard care se bazează, într-o mare măsură, pe „dialectul" SQL al IBM şi se întinde pe aproximativ 100 de pagini. Organizaţia Internaţională pentru Standarde (ISO) a adoptat propriul document, aproape identic cu ANSI SQL-86, pe care l-a publicat în 1987 ca *ISO 9075-1987 Database Language SQL.*

SQL-86 defineşte comenzile de bază ale SQL, inclusiv pentru crearea de tabele şi tabele virtuale (`CREATE TABLE, CREATE VIEW`), însă nu conţine opţiuni de modificare a structurii sau ştergere (`ALTER .../ DROP.,.`) şi nici comenzi pentru acordare şi revocare de drepturi utilizatorilor (`GRANT/REVOKE`)[22]. Lipsa facilităţilor privind definirea şi asigurarea restricţiilor referenţiale a generat discuţii aprinse şi critici vehemente, astfel încât rapid a fost publicat un set de specificaţii numit *Integrity Enhancement Feature* (elemente de ameliorare a integrităţii), prin care se pot defmi chei primare şi chei străine ca elemente componente ale schemei bazei de date.



Câteva abrevieri:
ISO - International Organisation lor Standardization ANSI - American National Standard Institute AFNOR - Association Frangaise de Normalisation OMG - Operaţional Management Committee DBSSG - Database Systems Study Group

PRIS-TG - Predictable Real-time Information Management Task Group **Figura 3.1.**

Structura organismelor de standardizare în domeniul bazelor de date

La trei ani de la publicarea SQL-86, prin revizuirea şi extinderea sa, se „naşte" SQL-89, care mai este denumit şi SQL-1 — ANSI X3.135-1989, respectiv ISO 9075:1989. Deşi recunoscut ca fundament al multor SGBDR-uri comerciale, şi SQL-1 şi-a atras numeroase critici. în plus, variantele comercializate de diferiţii producători, deşi esenţialmente asemănătoare, erau (şi sunt) incompatibile la nivel de detaliu. în afară de setul de specificaţii mai sus amintit, SQL-89 a inclus şi specificaţiile

22 Pentru detalii privind istoria SQL, vezi şi cuvântul înainte al lui Ken Jacobs la lucrarea IKreinesOOl s-- [Hernandez&ViescaOO], pp. 52-63.

pentru apelul comenzilor şi funcţiilor SQL din limbaje-gazdă, precum COBOL, Fortran şi C.

Pentru a umple golurile SQL-1, ANSI şi ISO au elaborat în 1992 versiunea SQL-2, ANSI X3.135-1992 *{Database Language SQL),* respectiv ISO/IEC 9075:1992, specificaţiile fiind prezentate la un nivel mult mai detaliat (dacă SQL-1 se întindea pe numai 100 de pagini, SQL-92 a fost publicat în aproape 600). Dintre numeroasele facilităţi aduse de SQL-92, merită amintite cu deosebire: joncţiunea externă `(OUTER JOIN),` atribute zi-oră şi de alte tipuri, raportare standardizată a erorilor, un set standard de tabele din catalog (dicţionarul de date), modificarea schemei bazei `(DROP, ALTER, GRANT, REVOKE),` SQL dinamic, modificări şi ştergeri referenţiale în cascadă, amânarea verificării restricţiilor, niveluri de consistenţă a tranzacţiilor etc.
Standardul SQL-92-defineşte trei niveluri de conformitate:

- *Entry* - intrare, de bază (opţiunile din SQL-89 corectate);
- *Intermediate* - intermediar (ce include aproximativ jumătate dintre facilităţi);
- *Full* - deplin.

Fiecare firmă îşi declară nivelul de conformitate al SGBD-ului în raport cu SQL-92. Spre exemplu, nucleul SQL din Oracle 8 este conform cu nivelul de bază (*entry*), dar, după declaraţiile producătorului, prezintă multe elemente suplimentare specifice celorlalte trei niveluri superioare.

Certificarea nivelului de conformitate cădea, până nu demult, în sarcina unui organism independent, *National Institute for Standards and Technology* (NIST), care utiliza *Federal Information Processing Standards* (FIPS), FIPS PUB 127-2. Din 1997 însă, FIPS şi-a declinat implicarea în activitatea de certificare[23].

Pe lângă ANSI, ale cărui standarde au cea mai largă audienţă, mai există şi alte organisme de standardizare SQL. X/Open este un grup de firme europene care a adoptat SQL ca nucleu al unei întregi serii de standarde menite să asigure realizarea unui mediu general pentru aplicaţii portabile, grefat pe sistemul de operare UNIX. IBM a avut un aport incontestabil la apariţia şi maturizarea SQL, fiind un producător cu mare influenţă în lumea SGBD-urilor, iar produsul său, DB2, este unul din standardele *defacto* ale SQL. ’

In 1989, un grup de producători de instrumente dedicate bazelor de date au format *SQL Access Group*, in vederea realizării conexiunilor dintre SGBDR-urile fiecăruia, pe baza unor specificaţii comune, din care un prim set a fost publicat în 1991 sub titulatura RDA (*Remote Database Access).* Specificaţiile RDA n-au reuşit să se impună pe piaţa SGBD-urilor. La insistenţele firmei Microsoft, SQL Access Group şi-a concentrat eforturile pentru elaborarea unei interfeţe-standard pentru SQL. Pe baza unui set de propuneri înaintat de companie, în 1992, au rezultat specificaţiile CLI *{Caii Level Interface).* Având drept reper CLI, Microsoft elaborează şi implementează în acelaşi an un set propriu, ODBC *(Open DataBase Conectivity),* care a devenit standard în materie de interfaţă SQL pentru accesarea diferitelor baze de date.

Cel mai recent standard este SQL-3, care a fost publicat în cea mai mare parte în iulie 1999. Complexitatea superioară faţă de precedesor este sugerată şi de numărul de pagini, aproape 2000 (faţă de 600 ale SQL-92). Scadenţa finalizării sale a fost repetat amânată. Principalele orientări ale SQL-3 vizează transformarea acestuia într-un limbaj complet, în vederea definirii şi gestionării obiectelor complexe şi persistente. Aceasta include:

- generalizare şi specializare,
- moşteniri multiple,
- polimorfism,
- încapsulare,
- tipuri de date definite de utilizator,
- suport pentru sisteme bazate pe gestiunea cunoştinţelor,

23 Vezi şi [Gorman97].

- expresii privind interogări recursive şi instrumente adecvate de administrare a datelor.

Defalcarea iniţială (1993) a standardului SQL operată de comitetele ANSI şi ISO a avut în vedere şapte componente[24]:

24 http://www.jcc.com/SQLPages/jccs_sql.htm.

Partea 1. *Cadru general.* Descrie fiecare parte a standardului şi conţine informaţii comune tuturor părţilor.

Partea 2. *Fundament.* Defineşte sintaxa şi semantica SQL în ceea ce priveşte definirea şi manipularea bazei de date, inclusiv opţiuni privind gestiunea tipurilor abstracte de date.

Partea 3. *SQL/CLI (Caii Level Interface*): un ansamblu de funcţii şi proceduri pentru conectarea bazelor de date prin SQL în medii multi-utilizator şi multi-platformă, ansamblu dezvoltat de SQL Access Group.

Partea 4. *SQL/PSM (Persistent Stored Modules):* specificaţii procedurale necesare în funcţii şi proceduri utilizator. în cele din urmă, elementele privind procedurile şi funcţiile, precum chestiuni legate de invocarea rutinelor, au fost transferate în partea a 2-a.

Partea 5. *SQL/Bindings:* include *Dynamic SQL* şi *Embedded SQL* din standardul SQL-92 (SQL-2). Se referă la modul în care SQL este inclus în limbajele de programare ne-obiectuale. Este de aşteptat ca în viitoarea versiune a standardului SQL această parte să fie mutată tot în partea a 2-a.

Partea 6. *SQL/XA:* specificaţii elaborate de X/Open şi dedicate platformei X Windows. Această parte a fost abandonată.

Partea 7. *SQL/Temporal,* adaugă noi facilităţi privind gestiunea timpului şi datei calendaristice în SQL.

La această structurare iniţială, între timp au mai fost adăugate şi alte părţi precum :

SQL/OLAP. Sunt descrise funcţiile şi operaţiunile utilizate pentru prelucrări analitice, fiind publicate ca amendament la standardul SQL-99[25].

Partea 8. *SQL/Objects - Extended Objects.* Vizează modul în care SGBD-urile relaţionale gestionează tipurile abstracte de date în aplicaţii. Nici această componentă nu mai există astăzi, fiind transferată în întregime în *Fundament.*

Partea 9. *SQL/MED (Management of Externai Data).* Defineşte câteva elemente adiţionale *Fundamentului* pentru accesarea unor surse (fişiere) de date non-SQL.

Partea 10: *SQL/OLB* (*Object Language Bindings).* Este inclusă numai în standardul ANSI şi definitivată din 1998; cuprinde specificaţii privitoare la includerea frazelor SELECT în limbajul Java, fiind corespondentă unui alt standard ANSI, SQLJ (Partea 0).

Partea 11: *SQL/Schemata.* Se referă la definirea şi extragerea informaţiilor privind schema bazei de date; este parte din *Fundament,* dar, în viitor, această parte va **fi de** sine stătătoare.

SQL Routines using the Java Programming Language. Definitivată în 1999 şi inclusă numai în standardul ANSI, această parte se referă la modul în care secvenţe de cod Java pot fi incluse în bazele de date SQL.

Partea 12: *SQL/Replication.* Lucrările au demarat în 2000, fiind vizată standardizarea comenzilor pentru replicarea bazelor de date.

Dintre ameliorările nelegate strict de lucrul cu obiecte, merită amintite cele referitoare la: declanşatoare (*triggere*) şi alte tipuri de proceduri stocate, suport pentru seturile de caractere naţionale, noi predicate în clauza `WHERE (FOR ALL, FOR SOME, SIMILAR TO)`, tabele virtuale actualizabile, roluri pentru definirea profilelor de securitate etc.

25 Vezi şi [FotacheOO - 4] şi [FotacheOO - 5].

în 1997, organizaţia X3 a ANSI a fost redenumită NCITS - National Committee for Information Technology Standards, iar comitetul care se ocupă de standardizarea SOL se numeşte acum ANSI NCITS-H2.

Se poate spune că, o dată cu standardul publicat în 1999, SQL iese din sfera relaţionalului şi a normalizării relaţiilor, aşa cum au fost formulate de Codd. în orice caz, SQL continuă să evolueze, iar organismele care-i guvernează standardizarea sunt preocupate de întâmpinarea cerinţelor pieţei bazelor de date.

Din perspectiva prezentei lucrări, obiectivul principal al SQL constă în a oferi utilizatorului mijloacele necesare formulării unei consultări numai prin descrierea rezultatului dorit, cu ajutorul unei aserţiuni (expresie logică), fără a fi necesară şi exphcitarea modului efectiv în care se face căutarea în BD. Altfel spus, *utilizatorul califică (specifică) rezultatul,* iar *sistemul se ocupă de procedura de căutare.*

Deşi este considerat, în primul rând, un limbaj de interogare, SQL este mult mai mult decât un instrument de consultare a bazelor de date, deoarece permite, în egală măsură:

- Definirea datelor
- Consultarea BD
- Manipularea datelor din bază
- Controlul accesului
- Partajarea bazei între mai mulţi utilizatori ai acesteia
- Menţinerea intergrităţii BD.

După Groff şi Weinberg, principalele atuuri ale SQL sunt[49]:

a) Independenţa de producător, nefiind o tehnologie proprietară
b) Portabilitate între diferite sisteme de operare
c) Este standardizat
d) „Filosofia sa se bazează pe modelul relaţional de organizare a datelor
e) Este un limbaj de nivel înalt, cu structură ce se apropie de limba engleză
f) Permite formularea de răspunsuri la numeroase interogări simple, ad-hoc neprevăzute iniţial
g) Constituie suportul programatic pentru accesul la BD
h) Permite multiple imagini asupra datelor bazei
i) Este un limbaj relaţional complet
j) Permite definirea dinamică a datelor, în sensul modificării structurii bazei chiar în timp ce o parte dintre utilizatori sunt conectaţi la BD k) Constituie un excelent suport pentru implementarea arhitecturilor client-server.

3.2. Tipuri de date şi comenzi principale SQL

50. [Celk99]

Înainte de prezentarea comenzilor SQL, ne ocupăm de categoriile principale de date ce pot fi stocate şi accesate într-o bază de date relaţională. Lista completă este cu mult mai lungă. în plus,

fiecare SGBD granularizează tipurile principale, astfel încât reprezentarea datelor să se facă într-un format cât mai apropiat de substanţa informaţiilor pe care le reflectă. Cu rea ştiinţă, au fost evitate, discret, tipurile definite de utilizator (UDT), în general, tot ce ţine de gestionarea obiectelor.

Specific nivelului de intrare *(entry)* al SQL-92 sunt următoarele categorii de date (împreună cu lungimea/precizia lor):

- `SMALLINT: întregi - scurte (4 poziţii, reprezentate pe 16 biţi),`
- `INTEGER` sau `INT:` întregi (9 poziţii, 32 biţi),
- `NUMERIC(p,5) sau DECIMAL(/?,s) sau DEC(p,s): reale, cu un total de` *p* `poziţii, din care 54a partea fracţionară,`
- `FLOAT: reale, virgulă mobilă (20 poziţii pentru mantisă),`
- `REAL:` real, virgulă mobilă (cu precizie mai mică decât `FLOAT,` dar la nivelul de intrare este identic),
- `DOUBLE PRECIS ION:` reale, virgulă mobilă, dublă precizie (30 de poziţii pentru mantisă),
- `CHAR(«)` sau `CHARACTER(«)`: şir de caractere de lungime *n* (max. 240),
- `VARCHAR(«)` sau `CHAR VARYING(«)` sau `CHARACTER VARYING(w):` şir de caractere de lungime variabilă (max. 254),
- `DATE: dată calendaristică,`
- `TIME: ora etc.,`
- `TIMESTAMP: an, lună, zi, ora, minutul, secunda, plus o fracţiune zecimală dintr-o secundă.`

Joe Celko tratează excelent fiecare categorie de dată, aşa cum este prezentă în SQL-92[50]. în ceea ce priveşte datele temporale, el operează o delimitare între:

- `evenimente, pentru care există tipurile DATE, TIME, TIMESTAMP,`
- intervale - perioade de timp dintre două evenimente (momente) - tipul `INTERVAL (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND)` şi
- perioade - sunt intervale cu un moment fix de început, pentru care este necesară o combinaţie de două câmpuri, fie `TIMESTAMP - TIMESTAMP,` fie `TIMESTAMP - INTERVAL.`

Cum lucrarea de faţă este consacrată nucleului SQL în trei dintre cele mai folosite SGBD-uri relaţionale, DB2 6.1 (IBM), Oracle 8 (Oracle) şi Visual FoxPro 6 (Microsoft), în tabelul 3.1 este prezentată o comparaţie sumară între acestea, din punctul de vedere al principalelor tipuri de date pe care le pot gestiona.

49. [Groff&Weinberg94], pp. 6-7.

Tabelul 3.1. Principalele tipuri de date utilizate în DB2, Oracle şi Visual FoxPro

DB2	Oracle	Visual FoxPro
CHARACTER(n)	CHAR(n)	CHARACTER(n)
CHAR(n)		
CHARACTER VARYING(n)	VARCHAR(n)	
CHAR VARYING (n)	VARCHAR2(n)	
VARCHAR(n)		
LONG VARCHAR(n)	LONG	
		LOGICAL
DECIMAL (p,s)	NUMBER(p,S)	NUMBER(p,s)
REAL (p,s)		
INTEGER	INTEGER	INTEGER
SMALLINT		
DOUBLE	FLOAT	DOUBLE
FLOAT	NUMBER	
DATE	DATE	DATE
TIME	DATE	DATETIME
TIMESTAMP	DATE (fără microsecunde)	

SQL este mai mult decât un limbaj de interogare, după cum am văzut în primul paragraf al acestui capitol. Acest lucru este ilustrat şi din evocarea principalelor familii de comenzi, fiecare familie vizând o sarcină specifică: interogare/manipulare, definirea datelor, controlul accesului, gestiunea tranzacţiilor etc. Principalele comenzi ale SQL care se regăsesc, într-o formă sau alta, în multe dintre SGBDR-urile actuale sunt:

Comandă	*Scop*
Pentru manipularea datelor	
SELECT	Extragerea datelor din bază Adăugarea de noi linii într-o tabelă
INSERT	Ştegerea de linii dintr-o tabelă Modificarea valorilor unor atribute
DELETE	
UPDATE	
	Adăugarea unei noi tabele în baza de date Ştergerea unei tabele din
Pentru definirea bazei de date	bază Modificarea structurii unei tabele Crearea unei tabele virtuale
CREATE TABLE	Ştergerea unei tabele virtuale
DROP TABLE	
ALTER TABLE	
CREATE VIEW	
DROP VIEW	

Pentru controlul accesului la BD	
`GRANT`	Acordarea unor drepturi pentru utilizatori
`REVOKE`	Revocarea unor drepturi pentru utilizatori
Pentru controlul tranzacţiilor	Marchează sfârşitul unei tranzacţii
`COMMIT`	Abandonează tranzacţia în curs
`ROLLBACK`	

în prezentul capitol, precum şi în următoarele, accentul va fi pus pe comenzile din primele două categorii. Comenzile SQL pentru controlul accesului şi al tranzacţiilor nu fac obiectul prezentei lucrări.

3.3. Crearea, modificarea şi ştergerea tabelelor. Restricţii

Crearea unei baze de date are o componentă tehnică pronunţată, mai ales în cazul serverelor de baze de date: Oracle, DB/2, SQL Server, Informix etc. Chiar dacă există instrumente de administrare care uşurează considerabil această activitate, crearea unei baze de date necesită cunoştinţe de administrare strict legate de produsul software de gestiune a bazei.

în cele ce urmează, vom eluda elementele tehnice/fizice (spaţiile-tabelă, segmentele de rollback, fişierele de date, procesele sistem etc.), iăsându-le în seama administratorilor BD; vom considera baza de date creată din punct de vedere fizic, astfel încât ne interesează numai crearea tabelelor, modificarea structurii lor şi declararea restricţiilor.

Crearea tabelelor şi declararea atributelor

Comanda SQL utilizată pentru crearea unei tabele este `CREATE TABLE`. Precizăm că este vorba de crearea structurii tabelei. Popularea cu înregistrări şi actualizarea ulterioară se realizează prin alte comenzi: `INSERT, UPDATE, DELETE`.

Formatul general al comenzii creatoare de tabele este:

```
   CREATE TABLE cnume tabelă> (clistă elemente din tabelă>) unde:
< listă elemente din tabelă > :     : = =
<element al tabelei> | <element al tabelei>, <listă elemente din
tabelă>
< element al tabelei > :   : = =
   < definiţie coloană > | < definiţie restricţie tabelă >
```

începem cu definiţiile coloanelor care cuprind o serie de opţiuni pentru declararea numelui, tipului, lungimii, precum şi restricţiilor:

```
< definiţie coloană > : : = =
   < nume coloană > < tip de dată >
   [< clauză valoarea implicită >]
   [< restricţie a coloanei >]
```

Pentru crearea unei tabele şi declararea atributelor acesteia, fără nici o referire la restricţii, comanda CREATE TABLE are, în cazul tabelei JUDEŢE, următoarea formă:

```
                    CREATE TABLE JUDEŢE (Jud CHAR(2) ,
      Judeţ VARCHAR(25),
      Regiune VARCHAR (15))
```

întrucât atributele acestei tabele sunt exclusiv de tip şir de caractere, s-au folosit opţiunile `CHAR` şi `VARCHAR. CHAR` este preferat pentru atributul Jud, deoarece indicativul auto al judeţului este format exclusiv din două litere (cu excepţia Bucureştiului). întrucât denumirile judeţului şi regiunii au lungime variabilă, pentru aceste atribute a fost preferat tipul `VARCHAR.`

Pentru ilustrarea şi altor două tipuri de date, prezentăm comanda de creare a tabelei FACTURI:

```
CREATE TABLE FACTURI (
   NrFact NUMERIC(8),
                              DataFact DATE,
   CodCl DECIMAL(6),
   Obs VARCHAR(50) )
```

`NrFact` şi `CodCl` sunt de tip numeric, în timp ce `DataFact` de tip dată calendaristică. Se mai cuvine de adăugat că unele atribute pot fi iniţializate cu o valoare implicită la adăugarea unei linii în tabela respectivă. Clauza este `DEFAULT:`

```
< clauză valoarea implicită > : : = =
   [ CONSTRAINT < nume restricţie > ]
DEFAULT < opţiune valoare implicită >
< opţiune valoare implicită > : : = = < literal > | < valoare
sistem > | NULL
```

Ultima comandă se poate rescrie astfel:

```
CREATE TABLE FACTURI (
   NrFact NUMERIC(8),
   DataFact DATE DEFAULT SYSDATE,
   CodCl DECIMAL(6) DEFAULT 1001,
   Obs VARCHAR(50) )
```

Ca urmare a clauzei `DEFAULT,` în orice linie adăugată în tabela FACTURI, valoarea implicită (dacă nu este specificată în comanda `INSERT)` a `DataFact` va fi data sistemului (data curentă), iar `CodCl` se va iniţializa cu valoarea 1001.

Mai trebuie amintit faptul că unele SGBD-uri, precum VFP, permit calculul valorilor implicite pe baza unei funcţii-utilizator (procedură stocată), iar altele, precum Oracle, permit folosirea secvenţelor în declanşatoare, utile mai ales pentru atribute-cheie, numerice.

Declararea restricţiilor

în SQL se pot declara toate tipurile de restricţii ale unei BDR: valori obligatorii (nenule), unicitate, restricţii referenţiale şi restricţii-utilizator. Continuând detalierea formulului general al comenzii `CREATE TABLE` în sintaxa SQL-92:

```
< restricţie a coloanei > :    : = = NOT NULL
    | < definiţie regulă de validare >
    | <   specificaţie privind unicitatea >
    | <   specificaţie referenţială >
```

Valori nenule

Unele atribute, obligatoriu cele din cheia primară, nu pot prezenta valori nule. Pentru aceasta, la crearea tabelei şi declararea atributului se foloseşte opţiunea `NOT NULL.` în unele produse (DB2, Oracle) trebuie indicate explicit atributele ce nu pot avea valori nule, în timp ce în Visual FoxPro este invers, trebuie declarate atributele susceptibile de valori NULL.

Şi între Oracle şi DB2 există o mică deosebire. în Oracle, pentru un atribut-cheie nu este obligatorie clauza `NOT NULL,` în timp ce în DB2 este. Iată noua formă a comenzii pentru crearea tabelei FACTURI:

```
CREATE TABLE FACTURI (
    NrFact NUMERIC(8) NOT NULL,
    DataFact DATE DEFAULT SYSDATE NOT NULL,
    CodCl DECIMAL(6) DEFAULT 1001 NOT NULL,
    Obs VARCHAR(50)  )
```

Cheie primară/unicitate

```
< specificaţie privind unicitatea > :     :  =  =  UNIQUE ]   PRIMARY
KEY
```

Cheia primară a unei relaţii este definită prin clauza `PRIMARY KEY,` plasată fie imediat după atributul-cheie, fie după descrierea ultimului atribut al tabelei. A doua variantă este întrebuinţată cu precădere atunci când cheia primară a tabelei este compusă.

Pentru atributele de tip cheie candidată se poate folosi clauza `UNIQUE,` care va asigura respectarea unicităţii valorilor. Iată câteva variante de folosire a clauzelor `PRIMARY KEY` şi `UNIQUE:`

```
CREATE TABLE FACTURI (
    NrFact NUMERIC(8) NOT NULL PRIMARY KEY,
    DataFact DATE DEFAULT SYSDATE NOT NULL,
    CodCl DECIMAL(6) DEFAULT 1001 NOT NULL,
    Obs VARCHAR(50)  )

CREATE TABLE JUDEŢE (
    Jud CHAR(2) PRIMARY KEY,
    Judeţ VARCHAR(25) NOT NULL UNIQUE,
    Regiune VARCHAR (15) )

CREATE TABLE LINIIFACT (
    NrFact NUMERIC(8) NOT NULL,
    Linie SMALLINT NOT NULL,
    CodPr NUMERIC(6) NOT NULL,
```

```
    Cantitate NUMERIC(10) NOT NULL,
    PretUnit NUMBER (12),
    PRIMARY KEY (NrFact, Linie),
    UNIQUE (NrFact, CodPr) )
```


Pentru tabela LINIIFACT, cheia primară este combinaţia de atribute `(NrFact, Linie)`, restricţia de unicitate pentru cuplul `(NrFact, CodPr)` înseamnă că se interzice ca, pe o factură, un produs să apară repetat.

Restricţii referenţiale

```
< specificaţie referenţială > : : = =
    [ CONSTRAINT < nume restricţie > ]
              REFERENCES < nume tabelă referenţiată >         [  (
          ccoloană referenţiată> ) ]
```

Declararea restricţiilor referenţiale se realizează utilizând clauza FOREIGN KEY. Astfel, pentru stabilirea legăturii LINIIFACT-FACTURI comanda de creare are forma:

```
CREATE TABLE LINIIFACT (
    NrFact NUMERIC(8) NOT NULL,
    Linie SMALLINT NOT NULL,
    CodPr NUMERIC(6) NOT NULL,
    Cantitate NUMERIC(10) NOT NULL,
    PretUnit NUMBER (12),
    PRIMARY KEY (NrFact, Linie),
    UNIQUE (NrFact, CodPr),
    FOREIGN KEY NrFact REFERENCES FACTURI(NrFact),
    FOREIGN KEY CodPr REFERENCES PRODUSE(CodPr) )
```

Este drept, şi următoarea variantă este corectă:

```
CREATE TABLE LINIIFACT (
    NrFact NUMERIC(8) NOT NULL REFERENCES FACTURI(NrFact),
    Linie SMALLINT NOT NULL,
    CodPr NUMERIC(6) NOT NULL REFERENCES PRODUSE(CodPr),
    Cantitate NUMERIC(10) NOT NULL,
    PretUnit NUMBER (12),
    PRIMARY KEY (NrFact, Linie),
    UNIQUE (NrFact, CodPr) )
```

în SQL-92 se poate specifica şi modul în care va fi păstrată integritatea bazei de date la ştergerea unei linii-părinte sau modificarea unei chei primare ce prezintă mregistran-copil.
S a interzice ştergerea unor facturi (înregistrări din FACTURI) pentru care exista SS corespondente în LINIIFACT şi, pe de altă parte, pentru ca la modificarea untn număr de factură `(NrFact)` în FACTURI sa se modifice automat, in cascada, toate liniile-copil din LINIIFACT, forma comenzii se schimbă în:

```
CREATE TABLE LINIIFACT (
    NrFact NUMERIC(8) NOT NULL,
    Linie SMALLINT NOT NULL,
    CodPr NUMERIC(6) NOT NULL,
    Cantitate NUMERIC(10) NOT NULL,
    PretUnit NUMBER (12),
    PRIMARY KEY (NrFact, Linie),
```

```
    UNIQUE (NrFact, CodPr),
    FOREIGN KEY NrFact REFERENCES FACTURI(NrFact)
    ON DELETE RESTRICT ON UPDATE
    CASCADE,
    FOREIGN KEY CodPr REFERENCES PRODUSE(CodPr)
    ON DELETE RESTRICT ON UPDATE
    CASCADE )
```

Restricţii-utilizator

în SGBD-urile actuale, restricţiile-utilizator, denumite şi restricţii de comportament sunt implementate sub forma regulilor de validare la nivel de camp *[field vahdation rule),* la nivel de înregistrare (*record validation rule*) sau, eventual, pot fi incluse in declanşatoare *(tnggere).*

```
< definiţie regulă de validare > :
    [ CONSTRAINT < nume restricţie >]
    CHECK ( < condiţie > )
```

Cu rezerva că aceste reguli pot avea forme complexe şi pot întrebuinţa elemente avansate de SQL şi/sau extensiile procedurale ale SQL în mediulrespectivprezentămin continuare o regulă de validare pentru atributul `DataFact` in tabela FACTURI şi o alta pentru atributul Linie în LINIIFACT.

```
CREATE TABLE FACTURI (
    NrFact NUMERIC(8) NOT NULL PRIMARY KEY,
    DataFact DATE DEFAULT SYSDATE NOT NULL
           CHECK (DataFact >=    '01/08/2000' AND DataFact <
    '31/12/2010',
    CodCl DECIMAL(6) DEFAULT 1001 NOT NULL
           REFERENCES CLIENŢI(CodCl) ON DELETE RESTRICT ON
    UPDATE CASCADE,
    Obs VARCHAR(50)  )

CREATE TABLE LINIIFACT (
    NrFact NUMERIC(8) NOT NULL,
    Linie SMALLINT NOT NULL CHECK (Linie > 0),
    CodPr NUMERIC(6) NOT NULL,
    Cantitate NUMERIC(10) NOT NULL,
    PretUnit NUMBER (12),
    PRIMARY KEY (NrFact, Linie),
    UNIQUE (NrFact, CodPr),
    FOREIGN KEY NrFact REFERENCES FACTURI(NrFact)
    ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY CodPr REFERENCES PRODUSE(CodPr)
    ON DELETE RESTRICT ON UPDATE CASCADE )
```

Ca urmare a clauzei `CHECK`, data unei facturi nu poate avea valori în afara intervalului `1` august `2000` — `31` decembrie `2010`, iar atributul `Linie` trebuie să prezinte valori întregi mai mari ca 0.

Crearea tabelelor şi declararea restricţiilor în Oracle 8

Ar fi o întreagă aventură dacă am încerca să discutăm formatul general al comenzii `CREATE TABLE`

în Oracle, şi aceasta jdeoarece, în afară de atribute şi restricţii, comanda permite specificarea unei serii întregi de parametri legaţi de definirea organizării tabelei, spaţiul-tabelă, caracteristicile de stocare, clustere, paralelism, partiţionare etc.

Astfel încât ne limităm discuţia la câteva consideremte generale. Fiecărei restricţii (cheie primară, unicitate, regulă la nivel de atribut etc.) i se poate da un nume, lucru util atunci când, la un moment dat (salvări, restaurări, încărcarea BD), vrem să dezactivăm una sau mai multe dintre acestea.

Pentru a uşura lucrul, se prefixează numele fiecărei restricţii cu tipul său:

- `pk_ (PRIMARY KEY) pentru cheile primare`
- `un_ (UNIQUE)` pentru cheile alternative
- `nn_ (NOT NULL)` pentru atributele obligatorii (ce nu pot avea valori nule)
- `ck_ (CHECK)` pentru reguli de validare la nivel de atribut
- `f k_ (FOREIGN KEY)` pentru cheile străine.

Am preferat tipul `NUMBER` pentru atributele numerice, deşi, cel puţin pentru coduri, era mai indicat tipul `INTEGER (PLS_INTEGER).` Pentru şiruri de caractere cu lungime variabilă, Oracle recomandă utilizarea tipului `VARCHAR2.` Lucrul cu atribute de tip `DATE` (dată calendaristică) necesită funcţia de conversie `TO_DATE` (). în Oracle nu există tipul de atribute `LOG I CAL` sau `BOOLEAN` (doar în PL/SQL o variabilă poate fi declarată `BOOLEAN`).

Atributele unei tabele pot avea definite valori implicite, însă acestea nu pot fi generate prin funcţii-utilizator, proceduri sau secvenţe. Declararea unei chei străine se realizează prin clauza `REFERENCES`. în Oracle (cel puţin până la versiunea 8) nu există clauză de actualizare în cascadă din tabelele-părinte în cele copil `(UPDATE CASCADE),` singura opţiune care se poate utiliza fiind `ON DELETE CASCADE,` utilă pentru ştergerea simultană a unei înregistrări-părinte şi a tuturor înregistrărilor-copil. Fireşte, prin mecanismul referenţial, opţiunile implicite sunt `ON UPDATE RESTRICT` şi `ON DELETE RESTRICT.`

în listing 3.1 este prezentat scriptul Oracle 8 pentru crearea tabelelor şi declararea restricţiilor bazei de date.

Listing 3.1. Script Oracle **8** de creare a tabelelor (şi declarare a restricţiilor)

```
DROP TABLE incasfact
DROP TABLE incasari ;
DROP TABLE liniifact
DROP TABLE facturi ;
DROP TABLE produse ;
DROP TABLE persclient
DROP TABLE persoane ;
DROP TABLE clienţi ;
DROP TABLE localitati
DROP TABLE judeţe ;

                CREATE TABLE judeţe ( jud CHAR(2)
     CONSTRAINT pk_judete PRIMARY KEY
     CONSTRAINT ck_jud CHECK (jud=LTRIM(UPPER(jud) ))
   , judeţ VARCHAR2(25)
     CONSTRAINT un_judet UNIQUE CONSTRAINT nn_judet
     NOT NULL CONSTRAINT ck_judeţ CHECK
          (judet=LTRIM(INITCAP(judeţ))
   ), regiune VARCHAR2(15)
     DE FAU LT 'Moldova'
     CONSTRAINT ck_regiune CHECK (regiune IN
        ('Banat', 'Transilvania', 'Dobrogea',
```

```
        'Oltenia',
        'Muntenia','Moldova'))
   );
CREATE TABLE localitati (
   codpost CHAR(5)
      CONSTRAINT pk_localitati PRIMARY KEY
      CONSTRAINT ck_codpost CHECK
   (codpost=LTRIM(codpost)), loc VARCHAR2(25)
      CONSTRAINT nn_loc NOT NULL
      CONSTRAINT ckJLoc CHECK (loc=LTRIM(INITCAP(loc))),
   jud CHAR(2)
                              DEFAULT 'IS'
      CONSTRAINT fk_localitati_jud REFERENCES judeţe(jud)
   );
               CREATE TABLE clienţi ( codcl NUMBER(6)
      CONSTRAINT pk_clienti PRIMARY KEY CONSTRAINT
   ck_codcl CHECK (codcl > 1000), dencl VARCHAR2(30)
      CONSTRAINT ck_dencl CHECK (SUBSTR(dencl,1,1) =
         UPPER(SUBSTR(dencl,1,1)))
   , codfiscal CHAR(9)
      CONSTRAINT ck_codfiscal CHECK
         (SUBSTR(codfiscal,1,1) =
         UPPER(SUBSTR(codfiscal,1,1))),
```

```
   adresa VARCHAR2(40)
      CONSTRAINT ck_adresa_clienti CHECK
        (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))) codpost
   CHAR(5)
      CONSTRAINT fk_clienti_localitati REFERENCES
   localitati(codpost), telefon VARCHAR2(10)
   );
CREATE TABLE persoane ( cnp
   CHAR(14)
      CONSTRAINT pk_persoane PRIMARY KEY CONSTRAINT ck_cnp
   CHECK (cnp=LTRIM(UPPER(cnp))) nume VARCHAR2(20)
      CONSTRAINT ck_nume CHECK (nume=LTRIM(INITCAP(nume))) prenume
   VARCHAR2(20)
      CONSTRAINT ck_prenume CHECK
        (prenume=LTRIM(INITCAP(prenume))), adresa
   VARCHAR2(40)
      CONSTRAINT ck_adresa_persoane CHECK
        (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))) sex
   CHAR(1) DEFAULT 'B'                                    '
      CONSTRAINT ck_sex CHECK (sex IN ('F','B')) codpost CHAR(5)   '
                                          '
      CONSTRAINT fk_persoane_localitati REFERENCES
   localitati(codpost), telacasa VARCHAR2(10), telbirou
   VARCHAR2(10), telmobil VARCHAR2(10), email VARCHAR2(20)
   );
CREATE TABLE persclienti ( cnp
   CHAR(14)
      CONSTRAINT fk_persclienti_persoane REFERENCES persoane(cnp),
   codcl NUMBER(6)
      CONSTRAINT fk_persclienti_clienti REFERENCES clienţi(codcl),
   funcţie VARCHAR2(25)
      CONSTRAINT ck_functie CHECK (SUBSTR(funcţie,1 1) =
        UPPER(SUBSTR(funcţie,1,1))),
   CONSTRAINT pk_persclienti ^ PRIMARY KEY (cnp,
   codcl, funcţie)

CREATE TABLE produse ( codpr
   NUMBER(6)
      CONSTRAINT pk_produse PRIMARY KEY CONSTRAINT ck_codpr CHECK
   (codpr > 0), denpr VARCHAR2(30) CONSTRAINT ck_denpr
   CHECK (SUBSTR(denpr,1,1) = UPPER(SUBSTR(denpr,1,1))) um
   VARCHAR2(10),                                          '
   grupa VARCHAR2(15)
   CHECK (SUBSTR(grupa,1,1) = UPPER (SUBSTR(grupa,1,1))) procTVA
   NUMBER(2,2) DEFAULT .22 );
               CREATE TABLE facturi ( nrfact NUMBER(8)
     CONSTRAINT pk_facturi PRIMARY KEY, datafact DATE
   DEFAULT SYSDATE
     CONSTRAINT ck_datafact CHECK (datafact >=
        TO_DATE('01/08/2000','DD/MM/YYYY')
     AND datafact <=
        TO_DATE('31/12/2010','DD/MM/YYYY')), codcl
   NUMBER(6)
       CONSTRAINT fk_facturi_clienti REFERENCES
                                clienţi(codcl) ,
                           Obs VARCHAR2(50)
   );
              CREATE TABLE liniifact ( nrfact NUMBER(8)
     CONSTRAINT fk_liniifact_facturi REFERENCES
```

```
   facturi(nrfact), linie NUMBER(2)
      CONSTRAINT ck_linie CHECK (linie >0), codpr NUMBER(6)
   CONSTRAINT fk_liniifact__produse REFERENCES produse(codpr),
   cantitate NUMBER(10), pretunit NUMBER (12),
   CONSTRAINT pk_liniifact PRIMARY KEY (nrfact,linie)
   );
               CREATE TABLE incasari ( codinc NUMBER(8)
      CONSTRAINT pk_incasari PRIMARY KEY, datainc DATE DEFAULT
   SYSDATE
      CONSTRAINT ck_datainc CHECK (datainc >=
         TO_DATE('01/08/2000','DD/MM/YYYY')
      AND datainc <=
         TO_DATE('31/12/2010','DD/MM/YYYY')), coddoc
   CHAR(4)
      CONSTRAINT ck_coddoc
         CHECK(coddoc=UPPER(LTRIM(coddoc))), nrdoc
   VARCHAR2(16), datadoc DATE DEFAULT SYSDATE - 7
   CONSTRAINT ck_datadoc CHECK
         (datadoc >= TO_DATE('01/01/2000','DD/MM/YYYY') AND datadoc
      <=
         TO_DATE('31/12/2010','DD/MM/YYYY'))
   );
CREATE TABLE incasfact (
   codinc NUMBER(8) CONSTRAINT fk_incasfact_incasari REFERENCES
   incasari(codinc) , nrfact NUMBER(8) CONSTRAINT
   fk_incasfact_facturi REFERENCES facturi(nrfact), transa
   NUMBER(16) NOT NULL,
   CONSTRAINT pk_incasfact PRIMARY KEY (codinc, nrfact) );
```

Să detaliem câteva dintre restricţiile declarate. Astfel, în tabela JUDEŢE:

- atributul `Jud` este cheie primară (restricţia `pk_j udete);`
- valorile lui `Jud` sunt exclusiv majuscule şi nu conţin spaţii la început `(ck_jud),` scop pentru care s-a folosit clauza `CHECK` şi funcţiile `LTRIM` şi `UPPER;`
- `Judeţ` este cheie alternativă `(un_j udet);`
- `Judeţ` nu poate avea valori nule `(nn_j udet);`
- iniţiala (sau, după caz, iniţialele) denumirii judeţului — `Judeţ` — este majusculă; restul sunt litere mici `(ck_j udet);`
- valoarea implicită pentru `Regiune` este ‘Moldova’;
- `Regiune` nu poate avea decât una dintre valorile: 'Banat', 'Transilvania', ’Dobrogea', 'Oltenia', 'Muntenia', 'Moldova' `(ck_regiune).`

Crearea tabelelor şi declararea restricţiilor în DB2

în afară de micile diferenţe legate de tipurile de date şi lucrul cu atribute de tip dată calendaristică, se cuvine menţionat faptul că în DB2, spre deosebire de Oracle, atributele din alcătuirea cheii primare trebuie declarate explicit ca fiind nenule (NOT `NULL).` Scriptul (ceva mai modest) de creare a tabelelor bazei de date este prezentat în listingul 3.2.

Listing 3.2. Crearea tabelelor în DB2

```
DROP TABLE incasfact ;
DROP TABLE incasari ;
DROP TABLE liniifact ;
DROP TABLE facturi ;
DROP TABLE produse ;
```

```
DROP TABLE persclienti ;
DROP TABLE persoane ;
DROP TABLE clienţi ;
DROP TABLE localitati ;
DROP TABLE judeţe ;
```



```
CREATE TABLE judeţe (
   jud CHAR(2) NOT NULL
     CONSTRAINT pk_jud PRIMARY KEY
     CONSTRAINT ck_jud CHECK (jud=LTRIM(UPPER(jud))), judeţ
  VARCHAR(25)
     CONSTRAINT un_judet UNIQUE ■ CONSTRAINT nn_judet NOT
  NULL, regiune VARCHAR(15)
      DEFAULT 'Moldova'
       CONSTRAINT ck_regiune CHECK (regiune IN
         ('Banat', 'Transilvania', 'Dobrogea',
  'Oltenia', 'Muntenia', 'Moldova'))
  );
CREATE TABLE localitati (
  codpost CHAR(5) NOT NULL
     CONSTRAINT pk_loc PRIMARY KEY
     CONSTRAINT ck_codpost CHECK (codpost=LTRIM(codpost)), loc
  VARCHAR(25)
     CONSTRAINT nn_loc NOT NULL, jud CHAR(2)
     DEFAULT 'IS'
     CONSTRAINT fk_loc__jud REFERENCES judeţe (jud)
  );
             CREATE TABLE clienţi (
                        codcl DECIMAL(6) NOT NULL
     CONSTRAINT pk_clienti PRIMARY KEY CONSTRAINT ck_codcl CHECK
  (codcl > 1000), dencl VARCHAR(30)
     CONSTRAINT ck_dencl CHECK (SUBSTR(dencl,1,1) =
        UPPER(SUBSTR(dencl,1,1))), codfiscal
  CHAR(9)
     CONSTRAINT ck_codfiscal CHECK (SUBSTR(codfiscal,1,1) =
  UPPER(SUBSTR(codfiscal, 1, 1) )) , adresa VARCHAR(40)
     CONSTRAINT ck_adresa_clienti CHECK
        (SUBSTR (adresa, 1,1) = UPPER (SUBSTR (adresa, 1,1) ).) ,
  codpost CHAR(5)
     CONSTRAINT fk_cl_loc REFERENCES localitati(codpost), telefon
  VARCHAR(IO)
  );
CREATE TABLE persoane ( cnp CHAR(14)
                            NOT NULL
     CONSTRAINT pk_persoane PRIMARY KEY CONSTRAINT ck_cnp CHECK
  (cnp=LTRIM(UPPER(cnp))), nume VARCHAR(20), prenume VARCHAR(20),
  adresa VARCHAR(40)
     CONSTRAINT ck_adresa_persoane CHECK
        (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))), sex CHAR(1)
  DEFAULT 'B'
     CONSTRAINT ck_sex CHECK (sex IN ('F','B')), codpost CHAR(5)
     CONSTRAINT fk_pers_loc REFERENCES localitati(codpost), telacasa
  VARCHAR(IO), telbirou VARCHAR(IO), telmobil VARCHAR(IO), . email
  VARCHAR(20)
  );
           CREATE TABLE persclienti ( cnp CHAR(14) NOT NULL
     CONSTRAINT fk_perscl_pers REFERENCES persoane(cnp), codcl
  DECIMAL(6) NOT NULL
     CONSTRAINT fk_perscl_cl REFERENCES clienţi(codcl), funcţie
  VARCHAR(25) NOT NULL '
```

```
    CONSTRAINT ck_functie CHECK (SUBSTR(funcţie,1,1) =
       UPPER(SUBSTR(funcţie,1,1))),
                     CONSTRAINT pk_persclienti
    PRIMARY KEY (cnp, codcl, funcţie)
 );
```

```
CREATE TABLE produse (
codpr DECIMAL(6) NOT NULL
   CONSTRAINT pk_produse PRIMARY KEY CONSTRAINT ck_codpr CHECK
(codpr > 0), denpr VARCHAR(30) CONSTRAINT ck_denpr
   CHECK (SUBSTR(denpr,1,1) = UPPER(SUBSTR(denpr,1,1))), um
VARCHAR(10), grupa VARCHAR(15)
   CHECK (SUBSTR(grupa,1,1) = UPPER(SUBSTR(grupa,1,1))), procTVA
DECIMAL(2,2) DEFAULT .22 );
CREATE TABLE facturi (
nrfact DECIMAL(8) NOT NULL
   CONSTRAINT pk_facturi PRIMARY KEY, datafact DATE DEFAULT CURRENT
DATE,
   CONSTRAINT ck_datafact CHECK (datafact >= '8/1/2000'
     AND datafact <= '12/31/2010'), codcl
DECIMAL(6)
   CONSTRAINT fk_fact_cl REFERENCES clienţi(codcl) ,
Obs VARCHAR(50)
);
CREATE TABLE liniifact (
nrfact DECIMAL(8) NOT NULL
   CONSTRAINT fk_lf_fact REFERENCES facturi(nrfact), linie
DECIMAL(2) NOT NULL
   CONSTRAINT ck_linie CHECK (linie > 0), codpr DECIMAL(6)
CONSTRAINT fk_lf_prod REFERENCES produse(codpr), cantitate
DECIMAL(IO), pretunit DECIMAL (12),
CONSTRAINT pk_liniifact PRIMARY KEY (nrfact,linie)
);
CREATE TABLE incasari (
codinc DECIMAL(8) NOT NULL
   CONSTRAINT pk_incasari PRIMARY KEY, datainc DATE
   CONSTRAINT ck_datainc CHECK (datainc >= '8/1/2000'
     AND datainc <= '12/31/2010'), coddoc
CHAR(4)
   CONSTRAINT ck_coddoc CHECK(coddoc=UPPER(LTRIM(coddoc))), nrdoc
VARCHAR(16), datadoc DATE
   CONSTRAINT ck_datadoc CHECK (datadoc >= '8/1/2000'
     AND datadoc <= '12/31/2010')
);
CREATE TABLE incasfact (
codinc DECIMAL(8) NOT NULL CONSTRAINT fk_if_inc REFERENCES
incasari(codinc) , nrfact DECIMAL(8) NOT NULL CONSTRAINT
fk_inc fact REFERENCES facturi(nrfact), transa DECIMAL(16)
NOT NULL,
CONSTRAINT pk_incasfact PRIMARY KEY (codinc, nrfact)
```

Nici în DB2 nu există, pentru restricţiile referenţiale, opţiunea de actualizare în cascadă a valorilor atributului-părinte în înregistrările-copil. Clauza ON UPDATE prezintă două variante: ON UPDATE NO ACTION şi ON UPDATE RESTRICT. Prima este periculoasă, deoarece lasă orfane înregistrările-copil.

Cât priveşte ştergerea unei înregistrări-părinte, se poate recurge la una din trei variante: ON DELETE NO ACTION (cea mai puţin recomandabilă), ON DELETE CASCADE şi ON DELETE SET NULL. După cum am văzut în primul capitol, în virtutea restricţiei referenţiale, este interzisă orice valoare nenulă într-o înregistrare-copil (pentru atributul cheie străină) care să nu se regăsească în tabela-părinte (tabela în care, de obicei, atributul de legătură este cheie primară sau candidată). Aceasta înseamnă că sunt admise înregistrările-copil orfane, dar numai la NULL-itatea cheii străine. Este ideea ultimei variante declarative - ON DELETE SET NULL: la ştergerea unei linii-părinte, în liniile copil atributele de legătură vor primi automat valoarea NULL.

Crearea tabelelor şi declararea restricţiilor în Visual FoxPro

Versiunea 3 şi noua denumire, Visual FoxPro, marchează momentul unei transformări radicale, din toate punctele de vedere, a produsului FoxPro. Noţiunea de bază de date devine una efectivă şi una de complezenţă. Restricţiile şi procedurile stocate sunt memorate în containerul bazei, forma „foxistă" a dicţionarului de date şi a mult mai complicatului catalog de sistem din serverele de bază de date.

Formatul general al comenzii CREATE TABLE în Visual FoxPro 3.0 este:

```
CREATE TABLE | DBF TableNamel [NAME LongTableName] [FREE]
   (FieldNamel FieldType [(nFieldWidth [, nPrecision])]
      [NULL | NOT NULL]
      [CHECK lExpressionl [ERROR cMessageTextl]]
      [DEFAULT eExpressionl]
      [PRIMARY KEY | UNIQUE]
      [REFERENCES TableName2 [TAG TagNamel]]
      [NOCPTRANS]
   [, FieldName2 ...]
      [, PRIMARY KEY eExpression2 TAG TagName2
      I, UNIQUE eExpression3 TAG TagName3]
      [, FOREIGN KEY eExpression4 TAG TagName4 [NODUP]
        REFERENCES TableName3 [TAG TagName5]]
      [, CHECK lExpression2 [ERROR cMessageText2]])
| FROM ARRAY ArrayName
```

Scurtă descriere a argumentelor:

`CREATE TABLE | DBF TableNamel`

TableNamel reprezintă numele tabelei ce urmează a fi creată. De remarcat că nu există nici o diferenţă între opţiunile TABLE şi DBF. Prima se adresează celor ataşaţi teoriei relaţionale, iar cea de-a doua este mai aproape de utilizatorul „tradiţional" al FoxPro (XBase-urilor, în general), pentru care o tabelă este un fişier cu extensia .DBF.

`NAME LongTableName`

Permite specificarea unui nume mai lung (până la 128 de caractere) pentru tabela creată. Pentru aceasta, este necesar ca baza de date să fie în prealabil deschisă, deoarece numele lungi sunt memorate în containerul asociat bazei (.DBC).

`FREE`

Indică faptul că tabela respectivă va fi independentă, deci nu va face parte din bază.

`(FieldNamel FieldType [(nFieldWidth [, nPrecision])]`

Permite declararea, pentru fiecare atribut (câmp) al tabelei, a numelui, tipului, lungimii şi, eventual, a numărului de poziţii pentru reprezentarea părţii fracţionare (zecimale).

`NULL`

Prin specificarea sa, atributul este autorizat să conţină valori nule (NULL).

`NOT NULL`

Previne apariţia valorilor nule pentru atributul respectiv. Automat, pentru atributele de tip cheie primară sau pentru care este utilizată opţiunea `UNIQUE,` nu se admit valori NULL.

`CHECK lExpressionl`

Serveşte la specificarea unei funcţii-utilizator de validare la nivel de atribut (câmp). Astfel, funcţia este verificată imediat după adăugarea unei noi înregistrări (linii) în tabelă. Dacă rezultatul evaluării funcţiei este *false,* se declanşează o eroare.

`ERROR cMessageTextl`

în cazul în care funcţia de validare de la nivelul atributului nu se respectă (adică „întoarce" valoarea logică `FALSE),` pe ecran apare mesajul `cMessageTextl.`

`DEFAULT eExpressionl`

Specifică valoarea implicită a atributului (în funcţie de tipul acestuia), valoare pe care o conţine acest câmp la adăugarea unei noi înregistrări în tabelă.

`PRIMARY KEY`

Declară atributul respectiv cheie primară, prin crearea unui index principal cu nume identic cu al atributului.

`UNIQUE`

Creează un „index candidat", altfel spus, declară acest atribut cheie alternativă. Se previne astfel apariţia a două valori identice în tabelă pentru acest câmp.

`REFERENCES TableName2 [TAG TagNamel]`

Permite definirea unei restricţii referenţiale prin crearea unei legături permanente între tabele. `TableName2` este tabeia-părinte a legăturii. Dacă nu se specifică `TAG TagNamel,` relaţia este stabilită prin intermediul indexului primar al tabelei `TableName2;` dacă acesta (indexul primar) nu există, se generează un mesaj de eroare. `TableName2` nu poate fi o tabelă independentă.

`NOCPTRANS`

Este utilă pentru câmpuri de tip şir de caractere şi memo, pentru a preveni conversia la un alt cod de pagină (cod page).

`PRIMARY KEY eExpression2 TAG TagName2`

Creează automat un index primar pentru tabela curentă. `eExpression2` este un câmp sau o combinaţie de câmpuri ale tabelei. Numele indexului poate conţine până la 10 caractere. Fireşte, clauza `PRIMARY KEY` nu poate apărea decât o singură dată într-o comandă `CREATE TABLE.`

`UNIQUE eExpression3 TAG TagName3`

Creează un index candidat. `eExpression3` desemnează orice câmp sau combinaţie de câmpuri ale tabelei, atribut/combinaţie care va îndeplini rolul de cheie alternativă. `TagName3` reprezintă numele indexului candidat al cărui nume este alcătuit tot din maximum 10 caractere. Pentru o aceeaşi tabelă pot fi creaţi mai mulţi indecşi candidaţi.

`FOREIGN KEY eExpression4 TAG TagName4 [NODUP]`

Are ca rezultat crearea unui index obişnuit (*regular*) pentru tabela curentă (cheia de indexare fiind `eExpression4,` iar numele acestuia `TagName4)` şi stabilirea unei relaţii permanente cu o tabelă-

părinte. Prin NODUP se poate crea un index străin candidat (prin analogie cu indecşii candidaţi la „postul" de index primar).

REFERENCES TableName3 [TAG TagName5]

Specifică numele tabelei-părinte implicate în legătura creată prin opţiunea FOREIGN KEY. Indexul TagName5 este cel prin care se realizează legătura dintre cele două tabele. Implicit este indexul primar al tabelei TableName3.

CHECK eExpression2 [ERROR cMessageText2]

Permite specificarea unei. restricţii (reguli de validare) la nivel de tabelă. cMessageText2 este mesajul afişat în cazul nerespectării restricţiei.

FROM ARRAY ArrayName

Permite crearea unei tabele pe baza datelor conţinute într-o variabilă de tip tablou.

Pentru comparaţie, în listingul 3.3 este prezentat programul pentru crearea în VFP 6 a bazei de date VÂNZĂRI.

Listing 3.3. Crearea tabelelor bazei de date în VFP6

```
CLOSE DATA CLOSE TABLES ALL
DELETE DATABASE VINZARI DELETETABLES
CREATE DATABASE vinzari
CREATE TABLE judeţe ( ; j ud CHAR(2)    ;
      PRIMARY KEY ;
      CHECK (jud=LTRIM(UPPER(jud))) ;
        ERROR 'Indicativul judeţului se scrie cu majuscule!',;
                            judeţ CHAR(25) ;
                                UNIQUE ;
                               NOT NULL ;
   CHECK (j udet=LTRIM(PROPER(j udet))) ;
      ERROR 'Prima litera din fiecare cuvint al'+CHR (13)+;
   'denumirii judeţului este majuscula; '+CHR(13)+; 'restul literelor
   sunt mici!', ; regiune CHAR(15) ;
      DEFAULT 'Moldova' ;
      CHECK (INLIST(regiune, 'Banat', 'Transilvania',; 'Dobrogea',
   'Oltenia', 'Muntenia', 'Moldova')) ; ERROR 'Regiunea poate avea o
   valoare numai din lista:'; +CHR(13)+'Banat, Transilvania,
   Dobrogea, ;
   Oltenia, ;Muntenia, Moldova' ;
   )
CREATE TABLE localitati ( ; codpost
   CHAR(5) ;
                              PRIMARY KEY ;
      CHECK (codpost=LTRIM(codpost)) ;
        ERROR 'Codul postai se scrie fara ; spatii la
   inceput !', ; loc CHAR(25) ;  ■
                               NOT NULL ;
      CHECK (loc=LTRIM(PROPER(loc) ) ) ;
        ERROR 'Prima litera din fiecare cuvint ;
          al'+CHR(13)+;
        'denumirii localitatii este majuscula;
        '+CHR(13)+;
        'restul literelor sunt mici!', ; jud CHAR(2) ;
      DEFAULT 'IS', ;
   FOREIGN KEY jud TAG jud REFERENCES judeţe TAG jud ;
   )
               CREATE TABLE clienţi ( ; codcl NUMBER(6) ;
                              PRIMARY KEY ;
      CHECK (codcl > 1000), ; dencl CHAR(30) ;
```

```
      CHECK(SUBSTR(dencl,1,1)=UPPER(SUBSTR(dencl, 1,1))), ; codfiscal
   CHAR(9) ;
   . NULL ;
        CHECK (SUBSTR(codfiscal,1,1)=;
             UPPER(SUBSTR(codfiscal,1,1))), ; adresa
   CHAR(40) ;
      NULL ;
      CHECK (SUBSTR(adresa,1,1) =;
        UPPER(SUBSTR(adresa,1,1))), ; codpost CHAR(5), ;
   telefon CHAR(10) ;
                                  NULL, ;
      FOREIGN KEY codpost TAG codpost ;
        REFERENCES localitati TAG codpost ;
   );
CREATE TABLE persoane ( ; cnp
   CHAR(14) ;
      PRIMARY KEY ;
      CHECK (cnp=LTRIM(UPPER(cnp))) ;
      ERROR 'Codul numeric personal se scrie fara ; spatii
   la inceput !', ; nume CHAR(20) ;
      CHECK (nume=LTRIM(PROPER(nume))) ;
        ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+; 'numelui
        este majuscula; '+CHR(13)+;
        'restul literelor sunt mici!', ; prenume CHAR(20) ;
      CHECK (prenume=LTRIM(PROPER(prenume))) ;
        ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
        'prenumelui este majuscula; '+CHR(13)+;
        'restul literelor sunt mici!', ; adresa CHAR(40) ;
      NULL ;
      CHECK (SUBSTR(adresa, 1, 1) = UPPER(SUBSTR(adresa,1,1)))     ;
        ERROR 'Prima litera din adresa este obligatoriu ; maj uscula
          !',           ;
   sex CHAR(1) DEFAULT 'B' ;
      CHECK (INLIST(sex,'F','B')) ;
        ERROR 'Atributul Sex poate avea valorile F ;
          (de la Femeiesc)'+CHR(13)+;
          'sau B (de la Bărbătesc) !', ;
   codpost CHAR(5), ; telacasa CHAR(10) NULL, ; telbirou
   CHAR(10) NULL, ; telmobil CHAR(10) NULL, ; email CHAR(20)
   NULL, ;
      FOREIGN KEY codpost TAG codpost ;
      REFERENCES localitati TAG codpost ;
   );
CREATE TABLE persclienti ( ; cnp
   CHAR(14), ; codcl NUMBER(6), ;
   funcţie CHAR(25) ;
      CHECK (SUBSTR(funcţie,1,1) = UPPER(SUBSTR(funcţie,1,1))) ERROR
      'Prima litera din funcţie este obligatoriu ; majuscula !', ;
   PRIMARY KEY cnp+STR(codcl,6)+functie TAG primaru, ;
      FOREIGN KEY cnp TAG cnp REFERENCES persoane TAG cnp, ; FOREIGN
      KEY codcl TAG codcl REFERENCES clienţi TAG codcl
   );
CREATE TABLE produse ( ; codpr
   NUMBER(6)       ;
      PRIMARY KEY ;
      CHECK (codpr >0) ;
        ERROR 'Codul produsului trebuie sa fie mai mare ; ca zero !',
   ; denpr CHAR(30) ;
      CHECK (SUBSTR(denpr,1,1) = UPPER(SUBSTR(denpr,1,1))) ;
```

```
      ERROR 'Prima litera din denumirea produsului ;
  este obligatoriu majuscula !', ; um CHAR(10), ; grupa
  CHAR(15) ;
     CHECK (SUBSTR(grupa,1,1) = UPPER(SUBSTR(grupa,1,1))) ;
       ERROR 'Prima litera din grupa este obligatoriu ;
  majuscula !', ; procTVA NUMBER(3,2) ;
     DEFAULT .22 ;
  )

             CREATE TABLE facturi ( ; nrfact NUMBER(8) ;
                    PRIMARY KEY, ; datafact DATE ;
     DEFAULT DATE() ;
        CHECK (BETWEEN (datafact, { ^2000/08/01}, {-'2010/12/31})
        ) ; ERROR 'Baza de date functioneaza in intervalul ;
        1 aug.2000 - 31 dec.2010 !', ; codcl NUMBER(6), ;
                          Obs CHAR(50) NULL, ;
         FOREIGN KEY codcl TAG codcl REFERENCES clienţi TAG codcl ;
  );
CREATE TABLE liniifact ( ;
  nrfact NUMBER(8), ; linie
  NUMBER(2) ;
     CHECK (linie >0) ;
       ERROR 'Atributul linie trebuie sa fie mai mare ; ca zero
  !', ; codpr NUMBER(6), ; cantitate NUMBER(10), ; pretunit
  NUMBER (12), ;
  PRIMARY KEY STR(nrfact,8)+STR(linie,2) TAG primaru, ;
     FOREIGN KEY nrfact TAG nrfact REFERENCES facturi TAG nrfact, ;
     FOREIGN KEY codpr TAG codpr REFERENCES produse TAG codpr ;
  )

              CREATE TABLE incasari ( ; codinc NUMBER(8) ;
     PRIMARY KEY, ;
  datainc DATE ;
       DEFAULT DATE() ;
       CHECK (BETWEEN(datainc, ;
            BETWEEN(datafact,{^2000/08/01}, {^2010/12/31})) ;
          ERROR 'Baza de date functioneaza in intervalul ;
          1 aug.2000 - 31 dec.2010 !', ; coddoc CHAR(4) ;
     CHECK(coddoc=UPPER(LTRIM(coddoc))) ;
       ERROR 'Codul documentului se scrie cu majuscule I', ;
  nrdoc CHAR(16), ; datadoc DATE ;
     DEFAULT DATE() - 7 ;
  CHECK (BETWEEN(datadoc,;
       BETWEEN (datafact, {''2000/01/01}, {^2010/12/31}) ) ;
       ERROR 'Data documentului trebuie sa fie intre ;
          1 ian.2000 si 31 dec.2010 !' ;
  );
CREATE TABLE incasfact ( ;
  codinc NUMBER(8), ;
     nrfact NUMBER (8),;
  transa NUMBER(16) ;
     NOT NULL, ;
  PRIMARY KEY STR(codinc,8)+STR(nrfact,8) TAG primaru, ;
     FOREIGN KEY codinc TAG codinc ;
     REFERENCES incasari TAG codinc, ;
     FOREIGN KEY nrfact TAG nrfact REFERENCES facturi TAG nrfact ;
  );
```

Nefiind un SGBDR de categoria grea, crearea bazei de date în VFP este un lucru uşor. fiind necesară o singură comandă, `CREATE DATABASE`. O opţiune interesantă la crearea tabelelor ţine de faptul că, în

afară de declararea restricţiilor, în VFP se poate stabili şi mesajul afişat la violarea restricţiei respective.

De asemenea, în VFP se indică explicit ce atribute *pot* avea valori nule (în Oracle şi DB2 se indică explicit cele care *nu pot* avea valori NULL). în plus, la atributele de tip numeric (reale), în numărul total de caractere o poziţie (distinctă) se contorizează pentru marca zecimală.

La declararea cheilor primare compuse, atributele componente trebuie concatenate. lucru ce atrage necesitatea conversiei celor de alt tip (numerice, dată calendaristică, logice) în şiruri de caractere. Declararea cheilor primare, alternative şi străine în VFP presupune crearea automată a indecşilor de tip `PRIMARY, CANDIDATE` sau `REGULAR.` Prin clauza `TAG` se poate da un nume la alegere indexului respectiv.

Din păcate, declararea cheilor străine nu înseamnă şi instituirea restricţiei referenţiale Aceasta este una dintre cele mai ciudate „găselniţe" VFP. Ca urmare, după crearea BD, fie trebuie „umblat" prin *Referenţial Integrity Builder* şi, astfel, grafic, instituite regulile pentru prezervarea referenţialităţii, fie trebuie create declanşatoare în acest scop (vezi capitolul 7).

Modificarea structurii tabelelor/restricţilor în Oracle, DB2 şi VFP

Schema unei baze de date reprezintă aspectul constant, invariabil sau, mai bine zis, puţin variabil în timp. O bună analiză şi proiectare a aplicaţiei (sistemului) diminuează riscul modificărilor de amploare în structura tabelor şi restricţii, conferind stabilitate bazei şi diminuând sensibil eforturile de întreţinere ulterioară instalării aplicaţiei.

Cu toate acestea, apariţia unui atribut nou, modificarea lungimii unui atribut sunt probleme de care orice analist/proiectant sau administrator/dezvoltator de aplicaţii şi BD s-a ciocnit cel puţin o dată (pe lună - e o glumă, fireşte!).

La aceste situaţii putem adăuga operaţiunile diverse - încarcărea bazei dintr-o altă aplicaţie, salvări, restaurări etc. - operaţiuni în care este necesară dezactivarea temporară a unor restricţii şi reactivarea lor ulterioară. în plus, destui practicieni obişnuiesc să creeze mai întâi toate tabelele (declararea atributelor) şi apoi să definească, printr-un script special, toate restricţiile bazei.

Astfel încât SQL prezintă o comandă dedicată modificării structurii bazei de date: `ALTER TABLE.`

```
ALTER TABLE <nume tabelă> <acţiune modificatoare a tabelei> unde:
< acţiune modificatoare a tabelei > : : = =
   ADD [COLUMN] <definiţie coloană>
   I ALTER [COLUMN] <nume coloană>
      <acţiune modificatoare a coloanei>
   | DROP [COLUMN] <nume coloană> <comportament la ştergere>
   I ADD <definiţie restricţie a tabelei>
   I DROP CONSTRAINT <nume restricţie>
      < comportament la ştergere >
```

Comportamentul la ştergere se referă la prohibirea sau propagarea în cascadă a ştergerii în înregistrările/tabelele-copil:

```
< comportament la ştergere > : : = = RESTRICT | CASCADE
```

Adăugarea

unui nou atribut

Adăugarea atributului `DataNast` (data naşterii) în tabela PERSOANE se realizează identic în toate cele trei produse, DB2/Oracle/VFP:

```
ALTER TABLE PERSOANE ADD DataNast DATE
```

Ştergerea

unui atribut

Curios sau nu, nici DB2, nici Oracle nu permit ştergerea unui atribut prin `ALTER TABLE,` în schimb, VFP este mai generos:

```
ALTER TABLE PERSOANE DROP COLUMN DataNast
```
Ba

chiar, în VFP un atribut se poate şi redenumi:

```
ALTER TABLE PERSOANE RENAME COLUMN DataNast TO DataNaşterii
```


Modificarea tipului/lungimii unui atribut

DB2 permite modificarea lungimii numai pentru atributele de tip `VARCHAR`. Pentru a creşte . dimensiunea atributului `Nume` în tabela PERSOANE la 21 de caractere se foloseşte:

```
ALTER TABLE PERSOANE ALTER Nume SET DATA TYPE VARCHAR(21)
```

în Oracle, modificarea tipului unui atribut se poate face astfel: din `CHAR` în `VARCHAR2` sau `VARCHAR;` din `VARCHAR2` sau `VARCHAR` în `CHAR,` dar numai dacă respectivul atribut conţine valoarea `NULL` în toate liniile tabelei sau dacă nu se modifică şi lungimea atributului. Lungimea poate fi mărită pentru orice atribut fără probleme; în schimb, micşorarea sa poate avea loc numai când valorile atributului respectiv sunt NULL. Pentru atingerea aceluiaşi scop ca în precendenta `ALTER TABLE` din DB2, în Oracle comanda are forma:

```
ALTER TABLE PERSOANE MODIFY (Nume VARCHAR2(21))
```

Visual FoxPro este mai puţin pretenţios la modificarea lungimii atributelor; acestea pot fi deopotrivă mărite sau micşorate; bineînţeles, la micşorare, trebuie avută în vedere trunchierea ce operează inevitabil.

```
ALTER TABLE PERSOANE ALTER COLUMN Nume CHAR(21)
```
dar şi

```
ALTER TABLE PERSOANE ALTER COLUMN Nume CHAR(17)
```

Adăugarea/modificarea valorii implicite

Declararea valorii implicite a unui atribut în DB2 este posibilă numai la crearea tabelei sau la adăugarea atributului.

în Oracle lucrurile sunt mai simple. Dacă se doreşte ca în liniile ce urmează a fi adăugate valoarea implicită a atributului `Sex` să fie 'F' (de la Femeiesc), comanda utilizată va `fi:`

```
ALTER TABLE PERSOANE MODIFY (Sex DEFAULT 'F')
```

Pentru anularea oricărei valori implicite se poate folosi:

```
ALTER TABLE PERSOANE MODIFY (Sex DEFAULT NULL)
```

Bineînţeles, atributele pentru care se declară `NULL` ca valoare implicită trebuie să „suporte" această (meta)valoare.

Şi VFP pemite modificarea valorilor implicite, prin:

```
ALTER TABLE PERSOANE ALTER COLUMN Sex SET DEFAULT 1F'
```

NULL şi neNULL ■

Pentru unele atribute poate fi instituită la crearea tabelei obligativitatea valorilor nenule. în timp, aceasta poate fi modificată într-un sens sau în celălalt. DB2 nu este prea flexibil în această privinţă, în schimb Oracle şi VFP da. Interzicerea valorilor nule pentru atributul `Sex` se realizează astfel:

- în Oracle:

```
ALTER TABLE PERSOANE MODIFY (Sex NOT NULL)
```

- în VFP:

```
ALTER TABLE PERSOANE ALTER COLUMN Sex NOT NULL
```
Invers, pentru a permite valori `NULL` pentru acelaşi atribut:

- în Oracle:

```
ALTER TABLE PERSOANE MODIFY (Sex NULL)
```

- în VFP: ~~....— .....—.....—

```
ALTER TABLE PERSOANE ALTER COLUMN Sex NULL A
```

dăugarea/anularea restricţiilor

Toate restricţiile: cheie primară - `PRIMARY KEY`, unicitate - `UNIQUE`, referenţială - `FOREIGN KEY`, de comportament - `CHECK` pot fi declarate ulterior creării tabelei şi,

bineînţeles, anulate la un moment dat. Spre exemplu, formatul general al comenzii pentru dezactivarea cheii primare, comun celor trei SGBD-uri este:

```
ALTER TABLE PERSOANE DROP PRIMARY KEY
```

De remarcat că, din cele trei produse, numai Oracle s-a „opus" vehement comenzii, motivând că există restricţii referenţiale declarate pe baza acestei chei primare. Pentru reinstituirea cheii primare a tabelei PERSOANE comanda are forma:

```
ALTER TABLE PERSOANE ADD PRIMARY KEY (CNP)
```

Analog, prin `ADD` şi `DROP` pot fi instituite/anulate şi celelalte tipuri de restricţii. Pentru- o mai simplă referire, este utilă folosirea clauzei `CONSTRAINT` (în DB2 şi Oracle) prin care se acordă un nume-utilizator fiecărei restricţii. Altminteri, pentru aflarea numelui restricţiei ce trebuie anulată este necesară consultatea catalogului sistem.

Ştergerea tabelelor

Comanda `DROP TABLE` şterge o comandă din baza de date. Sintaxa acesteia nu ridică probleme deosebite:

```
DROP <nume tabelă> < comportament la ştergere > unde:
< comportament la ştergere > : : = = RESTRICT | CASCADE
```

3.4. Inserarea, modificarea, ştergerea liniilor

SQL prezintă comenzi dedicate modificării conţinutului unei tabele, înţelegând prin aceasta trei acţiuni prin care se actualizează baza:

1. adăugarea de noi linii la cele existente în tabelă,
2. ştergerea unor linii,
3. modificarea valorii unui atribut.

Adăugarea unei linii

Comanda SQL de adăugare de noi linii este INSERT, care are, în modul de lucru cel mai puţin pretenţios, următorul format:

```
INSERT INTO tabelă [ (atributl, atribut2, ....)]
     VALUES (valoare_atributl, valoare__atribut2, ....)
```

In ceea ce priveşte scriptul de populare a tabelelor bazei de date cu valorile prezentate în primul capitol, diferenţele dintre DB2, Oracle şi VFP sunt minime, aşa încât nu prezentăm decât varianta Oracle din listingul 3.4.

Listing 3.4. Script Oracle 8 de populare a tabelelor


```
DELETE FROM incasfact ;
DELETE FROM incasari ;
DELETE FROM liniifact ;
DELETE FROM facturi ;
DELETE FROM produse ;
DELETE FROM persclienti DELETE FROM
persoane ;
DELETE FROM clienţi ;
DELETE FROM localitati DELETE FROM
judeţe ;

INSERT INTO judeţe VALUES  'IS'    'lasi', 'Moldova') ;
INSERT INTO judeţe VALUES  ,       'Vrancea',
INSERT INTO judeţe VALUES  'VN'    'Moldova')   'Neamţ',
INSERT INTO judeţe VALUES  ,       'Moldova')          ;
INSERT INTO judeţe VALUES    ('NT  'Suceava',
INSERT INTO judeţe VALUES      ',  'Moldova')  'Vaslui',
INSERT INTO localitati VALUES ('6600',  'Iasi', 'IS') ;
INSERT INTO localitati VALUES ('5725',  'Paşcani'  'IS') ;
INSERT INTO localitati VALUES ('6500',  'Vaslui', 'VS') ;
INSERT INTO localitati VALUES ('5300',  'Focşani'  'VN') ;
INSERT INTO localitati VALUES ('6400',  'Birlad',
INSERT INTO localitati VALUES ('5800',  'Suceava'  'SV') ;
INSERT INTO localitati VALUES ('5550 ',  'Roman',   NT' ) ;
INSERT INTO localitati VALUES (         'Timişoar  ',

 INSERT INTO clienţi VALUES (1001, 'Client 1 SRL', 'R1001',
                        '6600', NULL) ;
INSERT INTO clienţi (codcl, dencl, codfiscal, codpost, telefon.)
   VALUES (1002,'Client 2 SA', 'R1002',    '6600',  '032-212121') ;
INSERT INTO clienţi VALUES (1003, 'Client 3 SRL', 'R1003',
   'Prosperităţii, 22','6500','035-222222 ' ) ;
INSERT INTO clienţi (codcl, dencl, adresa, codpost)
   VALUES (1004, 'Client 4', 'Sapientei, 56',    '5725');
          INSERT INTO clienţi VALUES (1005, 'Client 5 SRL', 'R1005', NULL,
   '1900', '056-111111');
INSERT INTO clienti VALUES  (100   'Client 6 SA', 'R1006'
   'Pacientei, 33', '5550',  6,
INSERT INTO clienti VALUES (NULL   'Clien     SRL    'R1007'
   'Victoria Capitalismului,  )      t           -121212')
                           1007   1900',
INSERT INTO persoane VALUES ('CNP1', 'Ioan', 'Vasile',
   'I.L.Caragiale, 22', 'B','6600', '123456',   '987654',
   '094222222', NULL) ;
INSERT INTO persoane VALUES ( 'CNP2', 'Vasile', 'Ion', NULL, 'B',
   '6600', '234567', '876543', '094222223', 'Ion0a.ro') ;
INSERT INTO persoane VALUES ( 'CNP3',  'Popovici', 'Ioana',
   'V.Micle, Bl.I, Sc.B,Ap.2', 'F', '5725', '345678', NULL,
   '094222224', NULL) ;
```

```
INSERT INTO persoane VALUES ( 'CNP4', 'Lazar', 'Caraion',
'M.Eminescu, 42', 'B','6500',
'456789', NULL, '094222225', NULL) ;
INSERT INTO persoane VALUES ('CNP5',   'Iurea',  'Simion',
'I.Creanga, 44 bis', 'B',  '6500',     '567890', '543210',
NULL, NULL) ;
INSERT INTO  persoane VALUES  ('CNP6', 'Vase',    'Simona',
   'M.Eminescu, 13', 'F', '5725', NULL, '432109',
   '094222227', NULL) ;
INSERT INTO  persoane VALUES  ('CNP7', 'Popa',    'Ioanid',
   'I.Ion,-B1.H2, Sc.C, Ap.45', 'B',  '1900',    '789012',
   '321098', NULL, NULL) ;
INSERT INTO  persoane VALUES  ('CNP8', 'Bogacs', 'Ildiko',
   'I.V.Viteazu, 67', 'F', '5550', '890123',   '210987',
   '094222229', NULL) ;
INSERT INTO  persoane VALUES ( 'CNP9', 'loan',  'Vasilica',
   'Gării, B1.B4, Sc.A, Ap.l', 'F',
   '1900', '901234', '109876', '094222230', NULL) ;
```

```
INSERT   INTO   persclienti   VALUES  1001, 'Director general');
('CNP1',   INSERT   INTO   persclienti  1002, 'Director general');
VALUES    ('CNP2',    INSERT    INTO  1002, 'Sef aprovizionare');
persclienti VALUES ( 'CNP3', INSERT  1003, 'Sef aprovizionare');
INTO   persclienti   VALUES   ('CNP4',  1003, 'Director financiar')
INSERT   INTO   persclienti   VALUES  1004, 'Director general');
('CNP5',   INSERT   INTO   persclienti  1005, 'Sef aprovizionare');
VALUES   (   'CNP6',   INSERT   INTO  1006, 'Director
persclienti  VALUES  ('CNP7',  INSERT  financiar'),
INTO   persclienti   VALUES   (1CNP8',  1007, 'Sef aprovizionare');
INSERT   INTO   persclienti   VALUES  1','buc',     'Tigari',
('CNP9',                              .19)  2','kg',  'Bere',
INSERT INTO produse VALUES (1, 'Produs 0.19)     ;    3','kg',
INSERT INTO produse VALUES (2, 'Produs 'Bere',     0.19)     ;
INSERT INTO produse VALUES (3, 'Produs 4','1', 'Dulciuri',.19)
INSERT INTO produse VALUES (4, 'Produs 5','buc', 'Tigari', .19)
INSERT INTO produse VALUES (5, 'Produs
```

```
INSERT INTO facturi (nrfact, datafact, codcl)
  VALUES   (1111,   TO_DATE('01/08/2000','DD/MM/YYYY')   ,
  1001);
INSERT INTO facturi
  VALUES (1112, TO_DATE('01/08/2000','DD/MM/YYYY'), 1005,
  'Probleme cu transportul');
INSERT INTO facturi (nrfact, datafact, codcl)
  VALUES    (1113,    TO_DATE('01/08/2000','DD/MM/YYYY'),
  1002);
INSERT INTO facturi (nrfact, datafact, codcl)
  VALUES    (1114,    TO_DATE('01/08/2000','DD/MM/YYYY'),
  1006);
INSERT INTO facturi (nrfact, datafact, codcl)
  VALUES    (1115,    TO_DATE('02/08/2000','DD/MM/YYYY'),
  1001);
INSERT INTO facturi
  VALUES (1116, TO_DATE('02/08/2000','DD/MM/YYYY'), 1007,
  'Preţul propus iniţial a fost modificat');
INSERT INTO facturi (nrfact, datafact, codcl)
  VALUES    (1117,    TO_DATE('03/08/2000','DD/MM/YYYY'),
  1001);
```

```
INSERT INTO facturi (nrfact, datafact, codcl)
  VALUES    (1118,    TO_DATE('04/08/2000','DD/MM/YYYY'),
  1001);
INSERT INTO facturi (nrfact, datafact, codcl)
  VALUES    (1119,    TO_DATE('07/08/2000','DD/MM/YYYY'),
  1003);
```

```
INSERT INTO facturi (nrfact, datafact, codcl)
   VALUES (1120, TO_DATE('07/08/2000','DD/MM/YYYY'),
1001) INSERT INTO facturi (nrfact, datafact, codcl)
   VALUES (1121, TO_DATE('07/08/2000','DD/MM/YYYY'),
1004) INSERT INTO facturi (nrfact, datafact, codcl)
   VALUES (1122, TO_DATE('07/08/2000','DD/MM/YYYY'), 1005)
INSERT INTO liniifact VALUES (1111, 1, 1, 50, 10000)
INSERT INTO liniifact VALUES (1111, 2, 2, 75, 10500)
INSERT INTO liniifact VALUES (1111, 3, 5, 500  6500)
INSERT INTO liniifact VALUES (1112, 1, 2, 80, 10300)
INSERT INTO liniifact VALUES (1112, 2, 3, 40, 7 500)
INSERT INTO liniifact VALUES (1113, 1, 2, 100  9750)
INSERT INTO liniifact VALUES (1114, 1, 2, 70, 10700)
INSERT INTO liniifact VALUES (1114, 2, 4, 30, 15800)
INSERT INTO liniifact VALUES (1114, 3, 5, 700  6400)
INSERT INTO liniifact VALUES (1115, 1, 2, 150  9250)
INSERT INTO liniifact VALUES (1116, 1, 2, 125  9300)
INSERT INTO liniifact VALUES (1117, 1, 2, 100  10000)
INSERT INTO liniifact VALUES (1117, 2, 1, 100  9500)
INSERT INTO liniifact VALUES (1118, 1, 2, 30, 11000)
INSERT INTO liniifact VALUES (1118, 2, 1, 150  9300)
INSERT INTO liniifact VALUES (1119, 1, 2, 35, 10900)
INSERT INTO liniifact VALUES (1119, 2, 3, 40, 7000)
INSERT INTO liniifact VALUES (1119, 3, 4, 50, 14000)
INSERT INTO liniifact VALUES (1119, 4, 5, 750  6300)
INSERT INTO liniifact VALUES (1120, 1, 2, 80, 11200)
INSERT INTO liniifact VALUES (1121, 1, 5, 550  6400)
INSERT INTO liniifact VALUES (1121, 2, 2, 100  10500)
INSERT INTO incasari VALUES  1234,
   TO_DATE('15/08/2000','DD/MM/YYYY'),
   'OP','111',  TO_DATE('10/08/2000','DD/MM/YYYY')) ;
INSERT INTO incasari VALUES (1235,
   TO_DATE('15/08/2000','DD/MM/YYYY'),
   'CHIT', '222', TO_DATE('15/08/2000','DD/MM/YYYY'));
INSERT INTO incasari VALUES (1236,
   TO_DATE('16/08/2000','DD/MM/YYYY'),
      'OP',  '333', TO_DATE('09/08/2000','DD/MM/YYYY')) ;
INSERT INTO incasari VALUES (1237,
   TO_DATE('17/08/2000','DD/MM/YYYY'),
   'CEC', '444', TO_DATE('10/08/2000','DD/MM/YYYY'))
; INSERT INTO incasari VALUES (1238,
   TO_DATE('17/08/2000','DD/MM/YYYY'),
   'OP','555',  TO_DATE('10/08/2000','DD/MM/YYYY')) ;
INSERT INTO incasari VALUES (123 9,
   TO_DATE('18/08/2000','DD/MM/YYYY'),
   'OP','666',  TO_DATE('11/08/2000','DD/MM/YYYY')) ;
INSERT       INTO  incasfact VALUES(1234, 1111,
             5399625) ;
INSERT       INTO  incasfact VALUES(1234, 1118,
             1026375) ;
INSERT       INTO  incasfact VALUES(1235, 1112,
             487705) ;
```

```
      INSERT       INTO  incasfact VALUES(1236, 1117,
                   975410) ;
      INSERT       INTO  incasfact VALUES(1236, 1118,
                   1026375) ;
      INSERT       INTO  incasfact VALUES(1236, 1120,
                   731557) ;
INSERT INTO incasfact       VALUES    (1237, 1117, 975410) ;
INSERT INTO incasfact       VALUES    (1238, 1113, 1160250) ;
INSERT INTO incasfact       VALUES    (1239, 1117, 369680) ;
COMMIT ;
```

Ordinea valorilor din clauza `VALUES` trebuie să fie identică cu cea declarată la crearea (sau modificarea structurii) tabelelor. Modificarea este posibilă numai prin enumerarea, după numele tabelei, a atributelor ce vor primi valorile specificate.

Pentru tabelele CLIENŢI şi FACTURI au fost incluse în clauza `VALUES` mai puţine valori decât atributele tabelei. Este obligatorie, în aceste cazuri, precizarea atributelor care vor primi valorile. în Oracle şi DB2, restul, adică cele nespecificate, vor avea, pe liniile respective, valori `NULL`. în VFP, acestea sunt completate cu zero/spaţii (funcţia `EMPTY ()` întoarce `.T.`, în timp ce `ISNULL ()`. F.).

O problemă în VFP ţine de lucrul cu atribute de tip dată calendaristică. Listingul 3.5 prezintă, în acest sens, un extras din programul de populare a bazei de date, şianume comenzile `INSERT` pentru tabela FACTURI.

Listing 3.5. Constante de tip dată calendaristică în VFP

```
INSERT INTO  facturi(nrfact, datafact, codcl) VALUES (1111,;
  {^2000/08/01}, 1001)
INSERT INTO facturi VALUES (1112, {^2000/08/01}, 1005, ;
  'Probleme cu transportul', 0)
INSERT INTO  facturi(nrfact, datafact, codcl) ;
  VALUES (1113, {^2000/08/01}, 1002)
INSERT INTO  facturi(nrfact, datafact, codcl) ;
  VALUES (1114, {^2000/08/01}, 1006)
INSERT INTO  facturi(nrfact, datafact, codcl) ;
  VALUES (1115, {"'2000/08/02}, 1001)
INSERT INTO facturi VALUES (1116, {^2000/08/02}, ;
  1007, 'Preţul propus iniţial a fost modificat', 0)
INSERT INTO  facturi(nrfact, datafact, codcl) ;
  VALUES (1117, {^2000/08/03}, 1001)
INSERT INTO  facturi(nrfact, datafact, codcl) ;
  VALUES (1118, {^2000/08/04}, 1001)
INSERT INTO  facturi(nrfact, datafact, codcl) ;
  VALUES (1119, {^2000/08/07}, 1003)
INSERT INTO  facturi(nrfact, datafact, codcl) ;
  VALUES (1120, {^2000/08/07}, 1001)
INSERT INTO  facturi(nrfact, datafact, codcl) ;
  VALUES (1121, {^2000/08/07}, 1004)
INSERT INTO  facturi(nrfact, datafact, codcl) ;
  VALUES (1122, {^2000/08/07}, 1005)
```

Comanda INSERT permite şi adăugarea unor linii rezultate dintr-o consultare SQL (SELECT). Aici apar câteva diferenţe între VFP, Oracle şi DB2, pe care le vom discuta în capitolul 6.

Ştergerea liniilor

Comanda SQL pentru ştergerea uneia sau mai multor linii dintr-o tabelă este DELETE. Formatul general este:

```
DELETE
FROM nume-tabelă WHERE predicat
```

Din tabelă vor fi şterse toate liniile care îndeplinesc condiţia specificată în predicatul din clauza WHERE.

Exemplul 1
Să se elimine din tabela FACTURI factura cu numărul 1122.

```
DELETE FROM FACTURI WHERE NrFact = 1122
```

Varianta prezentată funcţionează în toate cele trei SGBD-uri. Dacă pentru această factură există întegistrări-copil în LINIIFACT sau INCASFACT, SGBD-ul ţine cont de opţiunea declarată la crearea tabelei. Dacă s-a specificat (implicit sau explicit) ON DELETE RESTRICT, ştergerea este interzisă. în cazul ON DELETE CASCADE sunt şterse, o dată cu linia din FACTURI, toate liniile-copil din LINIIFACT şi INCASFACT.

Exemplul 2

Să se elimine din baza de date toate judeţele din Moldova.

```
DELETE FROM JUDEŢE
    WHERE Regiune = 'Moldova'
```

Ca şi în cazul comenzii INSERT, pot fi şterse linii pe baza unei condiţii formulate printr-o subconsultare SQL, opţiune ce va fi prezentată în capitolul 6.

Modificarea valorilor unor atribute

Pentru a modifica valoarea unuia sau mai multor atribute pe una sau mai multe linii dintr-o tabelă, se foloseşte comanda UPDATE cu formatul general (simplificat):

```
UPDATE tabelă
SET atributl = expresiei [, atribut2= expresie2 ....]
WHERE predicat
```

Modificarea se va produce pe toate liniile tabelei care îndeplinesc condiţia formulată prin predicat.

Exemplul 3
Noul număr de telefon al clientului ce are codul 1001 este 032-313131. Să se opereze modificarea în baza de date.

Se actualizează atributul Telefon din tabela CLIENŢI:

```
UPDATE CLIENŢI
SET Telefon = '032-313131'
WHERE CodCl = 1001
```

Exemplul 4

In cadrul unei noi relaxări fiscale, se decide creşterea procentului TVA de la 19% la 22% pentru toate produsele.

Atributul modificat este ProcTVA din tabela PRODUSE, pe toate liniile:

```
UPDATE PRODUSE SET ProcTVA = .22
```

Capitolul 4

ELEMENTE DE BAZĂ ALE INTEROGĂRILOR SQL

Lucrarea de faţă este dedicată preponderent modalităţilor prin care, pornind de la o schemă relaţională, pot fi obţinute diverse informaţii dintr-o bază de date. Procesul se numeşte interogare, iar formularea unei interogări înseamnă redactarea unei fraze SELECT. Prezentul capitol tratează fundamentele redactării frazelor SELECT şi cele mai „populare" clauze ale acesteia.

4.1. Principalele clauze ale frazei SELECT

O frază SELECT are un format pe cât de simplu, pe atât de flexibil. Cele trei clauze principale sunt SELECT, FROM şi WHERE[26], dintre care numai primele două sunt obligatorii. Pentru început,"vomTâce o paralelă cu operatorii algebrei relaţionale prezentaţi în capitolul 2.

Selecţia şi proiecţia

Clauza SELECT corespunde operatorului proiecţie din algebra relaţională, fiind utilizată pentru desemnarea listei de atribute (coloanele) din rezultat. Clauza FROM este cealn care— sunt enumerate relaţiile din care vor fi extrase informaţiile aferente consultării. Prin WHERE' se desemnează predicatul selectiv al algebrei relaţionale, relativ la atribute ale relaţiilor care" apar în clauza FROM.

La modul general (şi simplist), o consultare simplă în SQL poate fi prezentată astfel:

SELECT CI, C2, . . . , Cn
frcSm **RÎ7** R2, **7777** .... **T5T~**
"WHEKHP P ---------- -

Prin exeeuţia unei fraze SELECT se obţine un rezultat de formă tabelară. Acesta poate fi o listă (text), o tabelă propriu-zisă sau o tabelă temporară (care, de obicei, nu poate fi actualizată), dar şi o tabelă derivată (imagine). Uneori rezultatul poate fi obţinut şi ca o variabilă masiv (tablou).

Ci - reprezintă coloanele (care sunt atribute sau expresii de atribute) rezultat;

Ftj - sunt relaţiile ce trebuie parcurse pentru obţinerea rezultatului;

P - este predicatul (condiţia) simplu sau compus ce trebuie îndeplinit de tupluri pentru a fi incluse în rezultat. " "..... "—————————————————————————————————-

Atunci când clauza WHERE este omisă, se consideră implicit că predicatul P are valoarea logică „adevărat", astfel încât vor fi incluse în rezultat toate liniile din tabela, sau produsul cartezian al tabelelor, enumerată/enumerate în clauza FROM.

Dacă, în locul coloanelor CI, C2, . . . Cn, apare simbolul *, rezultatul va fi alcătuit din toate coloanele (atributele) relaţiilor specificate în clauza FROM. De asemenea, atributele rezultatului preiau numele din tabela/tabelele specificate în FROM. Schimbarea numelui se realizează prin clauza AS.

în capitolul 1 era subliniată echivalenţa noţiunilor relaţie-tabelă. Conform restricţiei de unicitate,

26 La adresa http://w3.one.net/~jhoffman/sqltut.htm este un bun curs introductiv despre SQL.

într-o relaţie nu pot exista două linii identice. în SQL, tabela obţinută dintr-o consultare poate conţine două sau mai multe tupluri identice.

Spre deosebire de algebra relaţională, în SQL nu se elimină automat tuplurile identice fdublurilej din rezultat. Pentru aceasta este necesară utilizarea opţiunii DISTINCT:

```
SELECT DISTINCT CI, C2,                Cn
FRoM"*RT7 R2", . . ., "Km*
WfiERE P
```

în concluzie, o frază SELECT, în forma în care a fost prezentată, corespunde:

- unei selecţii algebrice (clauza WHERE P)
- unei proiecţii (SELECT Ci)
- unui produs cartezian (FROM - Rl ® R2 ® ... <g> Rm)

şi conduce la obţinerea unui rezultat cu *n* coloane, fiecare coloană fiind: un atribut din Rl, R2,..., Rm sau expresie calculată pe baza unor atribute din Rl, R2,..., Rm.

Vom trece în SQL câteva interogări din algebra relaţională, pe baza exemplelor din capitolul 2.

Exemplul 1 - selecţie

```
SELECT *
'—'FROM"TCl WHERE A > 20 AND C
> 20
```

Exemplul 2 - selecţie *(Care sunt judeţele din Moldova?)*

```
SELECT *
FROM JUDEŢE
WHERE Regiune = "Moldova"
```

Exemplul 3 - selecţie *(Care sunt facturile emise în perioada 2-5 august 2000?)*

Formatul standard al unei constante de tip datâ calendaristică este YYYY-MM-DD, aşa încât o interogare în SQL-92 poate avea forma:

```
SELECT *
FROM FACTURI
WHERE DataFact >= '2000/08/02' AND DataFact <= '2000/08/05'
```

Exemplul 4 - proiecţie

```
SELECT A,C FROM R
```

Exemplul 5 (Care sunt regiunile ţării preluate în bază?)

După cum aminteam mai sus, spre deosebire de algebra relaţională, SQL nu elimină automat dublurile. Tabela obţinută prin consultarea:

```
SELECT Regiune FROM JUDEŢE
```

are structura şi conţinutul identice cu R' (figura 2.11). Pentru a obţine răspunsul de formajabeki R (o regiune sa apară o_singură dată în răspuns), se foloseşte clauza DISTINCT:

```
SELECT DISTINCT Regiune FROM JUDEŢE
```

Exemplul 6 (Care sunt: codul, denumirea şi numărul de telefon ale fiecărui client?)

```
SELECT CodCl, DenCl, Telefon FROM CLIENŢI
```

Nu este necesară clauza DISTINCT, deoarece CodCl este cheia primară a tabelei CLIENŢI.

Exemplul 7 (Care este numărul de telefon al clientului Client 2 SA ?)

```
SELECT Telefon FROM CLIENŢI
WHERE DenCl = "Client 2 SA"
```

Exemplul 8 (Care sunt denumirile şi codurile poştale ale localităţilor (prezente în bază) din judeţele Iaşi (IS) şi Vrancea (VN) ?)

```
SELECT Loc, CodPost FROM LOCALITATI
WHERE Jud = 'IS' OR Jud = 'VN'
```

Reuniune, intersecţie, diferenţă, produs cartezian

Primii trei operatori asamblişti prezintă operatori SQL dedicaţi: UNION, INTERSECT. MINUS (EXTRACT), în timp ce produsul cartezian se calculează automat prin simpla enumerare a celor două tabele în clauza FROM.

Reuniunea

Un rezultat identic cu tabela R3 din figura 2.3 se obţine prin fraza SELECT următoare:

```
SELECT *
      FROM Rl
        UNION
            SELECT *
            FROM R2
```

La reuniunea a două tabele, SQL elimină automat liniile identice din rezultat. Dacă se doreşte preluarea *tuturor* liniilor celor două relaţii şi, implicit, apariţia de linii duplicate, se foloseşte clauza ALL astfel:

```
SELECT *
FROM Rl
    UNION ALL
            SELECT *
             FROM R2
```

Revenim la exemplul 8 din capitolul 2 (*Care sunt denumirile şi codurile poştale ale localităţilor (prezente în bază) din judeţele Iaşi (IS) şi Vrancea (VN) ?).*

Fraza SQL echivalentă soluţiei 2 (bazată pe reuniune) este:

```
SELECT Loc, CodPost FROM LOCALITATI WHERE Jud = 'IS'
    UNION SELECT Loc,
CodPost FROM
LOCALITATI WHERE Jud
= 'VN'
```

Intersecţia

Raportându-ne la exemplul din figura 2.3, echivalentul tabelei R4 se obţine în SQL prin:

```
SELECT *
FROM Rl
     INTERSECT
        SELECT *
         FROM R2
```

SQL-92 permite, ca şi în cazul reuniunii, prezenţa repetată în rezultat a unor linii (tupluri duplicate), bineînţeles, atunci când cele două relaţii asupra cărora se aplică intersecţia nu respectă restricţia de unicitate:

```
SELECT *
FROM Rl
     INTERSECT ALL
```

```
SELECT *
FROM R2
```

Dacă tuplul tl apare repetat şi în Rl, şi în R2, mai precis, de *n* ori în Rl şi de *m* ori în R2, în rezultatul operatorului INTERSECT tl apare o singură dată, în timp ce utilizând INTERSECT ALL tl va apărea de un număr de ori care este minimul dintre *n* şi *m.*

Pentru o primă ilustrare a utilizării intersecţiei, transcriem soluţia din algebra relaţională formulată la exemplul 9 (*Care sunt codurile produselor care apar simultan şi în factura 1111, şi în factura 1117?)'.*

```
SELECT CodPr FROM LINIIFACT WHERE NrFact = 1111 INTERSECT
```

```
SELECT      CodPr
FROM    LINIIFACT
WHERE   NrFact   =
1117
```

Exemplele de mai sus funcţionează în DB2 şi Oracle, nu însă şi în Visual FoxPro, care nu are implementat operatorul INTERSECT, astfel încât intersecţia trebuie realizată prin alte clauze şi operatori.

Diferenţa

Operatorul aşteptat ar fi MINUS. în standardul SQL-92, şi în câteva SGBDR-uri precum DB2, operatorul MINUS nu există, fiind substituit de EXCEPT, iar în alte SGBD-uri nu există nici unul, nici altul. Tabela R5 din figura 2.5, calculată prin expresia R1-R2, se obţine în SQL prin interogarea:

```
SELECT *
   FROM Rl
    EXCEPT
     SELECT *
     FROM R2
```

Oracle permite soluţia:

```
SELECT *
   FROM Rl
     MINUS
        SELECT *
        FROM R2
```

Ca şi în cazul reuniunii şi intersecţiei, eliminarea tuplurilor duplicate se face automat, iar repetarea lor se asigură prin EXCEPT ALL.

Modificăm enunţul exemplului 9 în următorul: *Care sunt codurile produselor care apar în factura 1111, dar nu apar în factura 1117?*

- ***Soluţia DB2:***

```
SELECT CodPr FROM
LINIIFACT WHERE
NrFact = 1111 EXCEPT
     SELECT      CodPr
     FROM    LINIIFACT
     WHERE   NrFact   =
     1117
```

- ***Soluţia Oracle:***

```
SELECT CodPr FROM
LINIIFACT WHERE
NrFact = 1111 MINUS
     SELECT      CodPr
     FROM    LINIIFACT
     WHERE   NrFact   =
     1117
```

Indiferent de variantă, rezultatul este o tabelă cu linie şi o coloană, valoarea fiind 5 (produsul ce are codul 5 apare în prima factură, dar nu şi în a doua).

Ca şi pentru INTERSECT, Visual FoxPro nu are implementat operator SQL pentru realizarea diferenţei a două relaţii.

Figura 4.1. Exemplu de coloană calculată

Pornind de la faptul că intersecţia nu este un operator fundamental, fraza SELECT ce realizează intersecţia se poate reformula cu ajutorul diferenţei astfel (fireşte, în Oracle se înlocuieşte EXCEPT cu MINUS):

```
SELECT *
FROM R1                                                    ,
   EXCEPT
      (SELECT *
      FROM R1
      EXCEPT
         SELECT *
         FROM R2)
```

Produsul cartezian

SQL nu pune la dispoziţie vreun operator special dedicat produsului cartezian. Nici nu este nevoie. Tabela R6 din figura 2.6 se obţine, pur şi simplu, prin enumerarea celor două relaţii în clauza FROM:

```
SELECT *
FROM R1, R2
```

Discuţia despre operatorii joncţiune şi diviziune se amână până la o dată ce va fi comunicată din timp. în continuare, prezentăm câteva facilităţi SQL privitoare la coloane calculate şi ordonarea liniilor în rezultatul interogării.

Coloane-expresii

O facilitate importantă în multe interogări SQL ţine de definirea, pe lângă atributele tabelelor, a unor coloane noi, pe baza unor expresii. Clauza AS permite denumirea coloanelor calculate sau redenumirea unor coloane ale tabelelor. Să luăm un exemplu banal: *Care este, pentru fiecare produs din factura 1111, codul, cantitatea, preţul unitar şi valoarea fară TVA ?*

```
SELECT CodPr, Cantitate, PretUnit, Cantitate * PretUnit
AS ValFaraTVA FROM LINIIFACT WHERE NrFact = 1111
```

Rezultatul interogării este prezentat în figura 4.1.

CODPR	CANTITATE	PRETUNIT	VALFARATVA
1	50	10000	500000
2	75	10500	787500
5	500	6500	3250000

A patra coloană este denumită ValFaraTVA, după cum a fost specificat în clauza AS. Valorile sale sunt determinate pe baza expresiei Cantitate * PretUnit.

Un alt tip de expresie este cel bazat pe concatenare, adică pe alipirea mai multor constante şi variabile într-o coloană nouă. Operatorul SQL pentru concatenare este alcătuit din două bare verticale (II). Spre exemplu, interogarea următoare produce rezultatul din figura 4.2.

```
SELECT 'Judeţul ' II Judeţ (|  ' se afla in ' || Regiune
   AS Exemplu_Concatenare
FROM JUDEŢE "
```

EXEMPLUJDON CATENARE
Judeţul Iasi se afla in Moldova _____
Judeţul Vrancea se afla in
Moldova Judeţul Neamţ se afla
in Moldova
Judeţul Suceava se afla in Moldova_
Judeţul Vaslui se afla in Moldova __
Judeţul Timiş se afla in Banat _____

Figura 4.2. Concatenarea unor literali şi atribute şir de caractere

Unele SGBD-uri, precum VFP, nu au implementat acest operator, folosind în acest scop unul specific (în cazul VFP este +).

în Oracle, se pot concatena direct, adică fară conversie prealabilă, literali, atribute şir de caractere, numerice etc. Alte SGBD-uri, precum DB2, necesită transformarea valorilor non-şir de caractere (atribute/constante numerice, dată calendaristică etc.), operaţiune realizată prin funcţia `CAST`.

Spre exemplu, în Oracle, interogarea următoare este funcţională:

```
SELECT   'Factura ' || NrFact || 'a fost emisa pe data '    ||
   DataFact AS Concatenare_Oracle
FROM FACTURI
```

în timp ce DB2 nu o suportă, şi asta nu neapărat din cauza faptului că am denumit coloana calculată `Concatenare__Oracle`. Folosind funcţia de conversie `CAST`, putem redacta următoarea variantă a interogării, mult mai pe gustul DB2:

```
SELECT   'Factura ' || CAST (NrFact AS CHAR(8)) ||
   ' a fost emisa pe data '  ||CAST (DataFact AS
       VARCHAR(10)) AS Concatenare_DB2
FROM FACTURI
```

Valorile atributului `NrFact` sunt transformate în şiruri de caractere de lungime fixă (8 poziţii), în timp ce ale `DataFact` vor fi convertite în şiruri de caractere de lungime variabilă (vorba vine, deoarece formatul datei este unitar pentru toate liniile rezultatului) de maximum 10 poziţii.

Visual FoxPro face notă discordantă, întrucât pentru conversia valorilor din numeric în şir de caractere se foloseşte funcţia STR (), iar diri'dăfa‘căiendaristică, după caz, DTOC () jau~DTOS ()7'

```
SELECT   'Factura '+STR(NrFact,8)+' a fost emisa pe data   '+;
   DTOC(DataFact) AS Concatenare_VFPl ;
FROM FACTURI
```

In finalul discuţiei despre coloanele-expresii, mai zăbovim preţ de câteva rânduri asupra expresiilor de tip dată calendaristică. Modurile în care au fost implementate aceste funcţiuni sunt foarte eterogene, de la SGBD la SGBD. Să presupunem că orice factură trebuie încasată în maximum două săptămâni. Pentru a afişa scadenţele, soluţia DB2 este:

```
SELECT NrFact AS Factura, DataFact AS Data_Fact urare,
     DataFact + 14 DAY AS Scadenta_Incasare
FROM FACTURI
```

FACTURA	DATA_F ACTU RAR E	SCADENTAJNCASARE
1111	2000-08-01	2000-08-15
1112	2000-08-01	2000-08-15
1113	2000-08-01	2000-08-15
1114	2000-08-01	'2000-08-15
1115	2000-08-02	2000-08-16
1116	2000-08-02	2000-08-16
1117	2000-08-03	2000-08-17
1118	2000-08-04	2000-08-18
1119	2000-08-07	2000-08-21
1120	2000-08-07	2000-08-21
1121	2000-08-07	2000-08-21
1122	2000-08-07	2000-08-21

Figura 4.3. O expresie

Data care ne interesează se obţine graţie expresiei `DataFact + 14 DAY`. într-o expresie de tip `DATE` sau `DATETIME`, imediat după număr, trebuie indicată semnificaţia acestuia: `YEAR (YEARS)`, `MONTH (MONTHS)`, `DAY (DAYS)`, sau, după caz, `HOUR (HOURS)`, `MINUTE (MINUTES)`, `SECOND (SECONDS)`, `MICROSECOND (MICROSECONDS)`.

Figura 4.1. Exemplu de coloană calculată

La aceeaşi problemă, Oracle şi Visual FoxPro permit soluţii mai simple (de data aceasta);

DataFact fiind un atribut de tip DATE, implicit se consideră 14 ca fiind numărul zilelor:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
     DataFact + 14 AS Scadenta_Incasare
FROM FACTURI
```

Dacă am presupune că scadenţa ar fi peste două luni de la facturare, interogările ar trebui modificate astfel:

- DB2:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
    DataFact + 2 MONTHS AS Scadenta_Incasare
FROM FACTURI
```

- Oracle:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
ADD__MONTHS (DataFact, 2) AS Scadenta încasare FROM
FACTURI                                    ~
```

- Funcţia Visual FoxPro corespondentă ADD_MONTHS este GOMONTH:

```
SELECT NrFact AS Factura, DataFact AS Data Facturare, ;
GOMONTH(DataFact,2) AS Scadenta încasare ;
FROM FACTURI                              "
```

Complicându-ne şi mai zdravăn, dorim să afişăm pentru fiecare factură ce dată va fi peste 1 an, două luni şi 25 de zile de la momentul emiterii.

- Soluţia DB2:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
     DataFact + 1 YEARS + 2 MONTHS + 25 DAYS AS
0_Data_Viitoare FROM FACTURI
```

- Soluţia Oracle necesită transformarea anilor în luni:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
ADD_MONTHS(DataFact,14)+15 AS 0_Data_Viitoare FROM FACTURI
```

- în VFP:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare, ;
   GQMONTH(DataFact,14)+15 AS 0_Data_Viitoare ;
FROM FACTURI
```

în ceea ce priveşte operaţiunile de adunare şi scădere între două date calendaristice, aici lucrurile se prezintă şi mai diferenţiat. Spre exemplu, interesează intervalul scurs între momentul curent şi cel al emiterii fiecărei facturi. Interogarea DB2:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
   CURRENT DATE - DataFact AS Timp_Scurs
FROM FACTURI
```

furnizează valorile din figura 4.4. Trebuie avut în vedere că interogarea a fost executată pe 5 februarie 2001.

FACTURA	DATA_F ACTU RAR E	TIMP SCURS
1111	2000-08-01	604
1112	2000-08-01	604
1113	2000-08-01	604
1114	2000-08-01	604
1115	2000-08-02	603
1116	2000-08-02	603
1117	2000-08-03	602
1118	2000-08-04	601
1119	2000-08-07	529
1120	2000-08-07	529
1121	2000-08-07	529
1122	2000-08-07	529

Figura 4.4. Intervalul dintre 05/02/2001 **(data execuţiei interogării) şi data fiecărei facturi** (DB2)

Rezultatul scăderii a două date calendaristice conţine numărul de ani, luni şi zile. în figura 4.4, valoarea 604 înseamnă 6 luni şi 4 zile, iar 529 - 5 luni şi 29 de zile. Bineînţeles că apelând la funcţii de conversie şi extragere se poate obţine un format de prezentare ceva mai agreabil.

în Oracle, avem nevoie de câteva artificii între data curentă (SYSDATE) şi data facturării. Funcţia MONTHS_BETWEEN calculează numărul lunilor cuprinse între două date calendaristice:

```
SELECT NrFact AS Factura, DataFact AS Data_Facturare,
    TRUNC(MONTHS_BETWEEN(TRUNC(SYSDATE), DataFact),0)
  AS Luni_Scurse,
ADD_MONTHS(TRUNC(SYSDATE), -
    TRUNC(MONTHS_BETWEEN(TRUNC(SYSDATE), DataFact),0))
    - DataFact AS Zile_Scurse FROM FACTURI
```

Interogarea generează un rezultat de forma celui din figura 4.5.

FACTURA	DAT A_F ACTURARE	LUNLSCURSE	ZILE_SCURSE
1111	01 -AUG-00	6	4
1112	01 -AUG-00	6	4
1113	01 -AUG-00	6	4
1114	01-AUG-00	6	4
1115	02-AUG-00	6	3
1116	02-AUG-00	6	3
1117	03-AUG-00	6	2
1118	04-AUG-00	6	1
1119	07-AUG-00	5	29
1120	07-AUG-00	5	29
1121	07-AUG-00	5	29
1122	07-AUG-00	5	29

Figura 4.5. Intervalul dintre 5 febr. 2001 (data execuţiei interogării) şi data fiecărei facturi (Oracle)

Opţiunea ORDER BY

Una dintre caracteristicile modelului relaţional este că nici ordinea atributelor, nici ordinea liniilor în relaţii nu prezintă importanţă din punctul de vedere al conţinutului informaţional. In practică însă, forma de prezentare a rezultatelor interogării este importantă. Spre exemplu, o listă a localităţilor este cu mult mai de folos decă se prezintă în ordine alfabetică. Ordonarea liniilor în rezultatul unei interogări este posibilă prin clauza ORDER BY.

Să se obţină lista localităţilor în ordine alfabetică.

```
SELECT *
FROM LOCALITATI
ORDER BY Loc
```

CODPOST	LOC	JUD
6400	Birlad	VS
5300	Focşani	VN
6600	Iasi	IS
5725	Paşcani	IS
5550	Roman	NT
5800	Suceava	SV
1900	Timişoara	TM
6500	Vaslui	VS

Figura 4.6. Localităţile ordonate alfabetic

Aranjarea se face implicit crescător (ASC). Prin opţiunea DESC, ordinea prezentării se inversează. în plus, se pot specifica mai multe coloane care să servească drept criterii suplimentare de ordonare. La valori egale ale primului atribut, intră în acţiune criteriul de „balotaj", care este al doilea atribut etc.

Să se obţină, în ordinea descrescătoare a indicativului judeţelor, lista localităţilor în ordinea crescătoare a denumirii.

```
SELECT Jud, Loc, CodPost
FROM LOCALITATI
ORDER BY Jud DESC, Loc ASC
```

Rezultatul este cel din figura 4.7. Ultimul indicativ de judeţ (e vorba de ordine

Rezultatul se prezintă ca în figura 4.6.

alfabetică) din tabela LOCALITĂŢI este VS (pentru Vaslui), deci primele localităţi afişate vor fi din acest judeţ: prin urmare, prima linie a rezultatului se referă la municipiul Bârlad, iar a doua la municipiul Vaslui.

JUD	LOC	CODPOST
VS	Birlad	6400
VS	Vaslui	6500
VN	Focşani	5300
TM	Timişoara	1900
SV	Suceava	5800
NT	Roman	5550
IS	Iasi	6600
IS	Paşcani	5725

Figura 4.7. Două criterii de ordonare

4.2. Operatorii BETWEEN, LIKE, IN

Pentru formularea predicatului de selecţie, SQL permite utilizarea, pe lângă „clasicii" >, >, <, <, =, *, şi a altor operatori, dintre care ne vom opri la: BETWEEN (între, cuprins între). LIKE (ca şi, la fel ca), IN (în), la care se adaugă IS NULL, ce va fi prezentat în capitolul 5.

Operatorul BETWEEN

Este util pentru definirea intervalelor de valori. Ne reîntoarcem în capitolul 2, la *exemplul 3 {Care sunt facturile emise în perioada 2-5 august 2000?).*

Pe lângă soluţia formulată anterior în acest capitol, vom prezenta noi variante DB2, Oracle şi VFP. Profităm de ocazie pentru a prezenta şi modul de vizualizare a rezultatelor cele trei SGBD-uri.

- Soluţie DB2 v6.1 - vizualizare rezultat - figura 4.8.

```
SELECT *
FROM FACTURI
WHERE DataFact BETWEEN '02/08/2000' AND '05/08/2000'
```

```
NRFACT DfiTflFflCT      CODCL     OBS

      1115. 08/02/2800    1001 .
      1116. 08/02/2000   1 007 . Preţul propus iniţial a Fost modificat
      1117. 88/03/2000    1001 .
      1118. 08/04/2000    1001 . -

  4 record(s) selected -
```

Figura 4.8. Operator BETWEEN - rezultat DB2 v6.1

- Soluţia Oracle - rezultat figura 4.9. Specificarea constantelor de tip dată calendaristică fac necesară în Oracle funcţia TO_DATE.

```
SELECT *                              .
FROM FACTURI
WHERE DataFact BETWEEN TO_DATEf02/08/2000','DD/MM/YYYY')
   AND TO_DATE('05/08/2000','DD/MM/YYYY') ;
```

```
   NRFACT  DATAFACT       CODCL  OBS

     1115  02-AUG-00       1001
     1116  02-AUG-00       1007  Preţul propus iniţial a fost modificat
     1117  03-AUG-00       1001
     1118  04-AUG-00       1001
```

Figura 4.9. Operator BETWEEN - rezultat Oracle 8 (SQL*Plus)

- Constantele de tip dată calendaristică trebuie încadrate de acolade în VFP. Ecranul de „răspuns" este cel din figura 4.10.

```
SELECT * ;
FROM FACTURI ;                          ....~  ~     ~~--- ^
WHERE DataFact BETWEEN {02/08/2000} AND {05/08/2000,}""
```

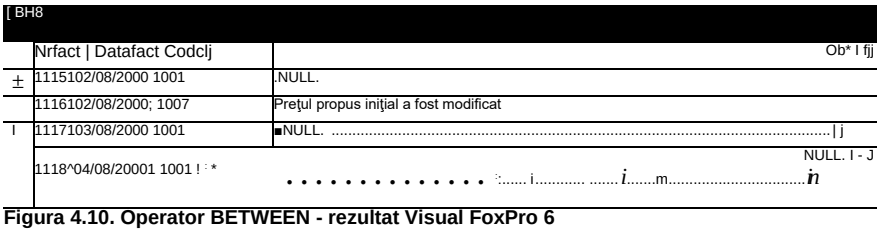


Figura 4.10. Operator BETWEEN - rezultat Visual FoxPro 6

Să se obţină, în ordinea judeţelor, lista localităţilor cu indicativul judeţului cuprins între IS (Iaşi) şi TM (Timiş).

```
SELECT Jud, Loc, CodPost FROM LOCALITATI
WHERE Jud BETWEEN 'IS' AND 'TM'
ORDER BY Jud, Loc
```

JUD	LOC	CODPOST
IS	Iasi	6600
IS	Paşcani	5725
NT	Roman	5550
SV	Suceava	5800
TM	Timişoara	1900

Figura 4.11. Operatorul BETWEEN aplicat atributelor şir de caractere

Din examinarea cu ochiul liber a celor trei figuri reiese că nici DB2 şi nici SQL*Plus (Oracle 8) nu afişează valorile `NULL` în clar, ci spaţii.

Operatorul LIKE

De multe ori, când se doreşte obţinerea unor informaţii din bază suntem în postura, oarecum ingrată, de a nu şti cu exactitate cum se numeşte un client sau un produs, sau, pur şi simplu, unei persoane îi cunoaştem numai unul dintre prenume etc. Sunt numai câteva situaţii pentru rezolvarea cărora a fost gândit operatorul `LIKE`.

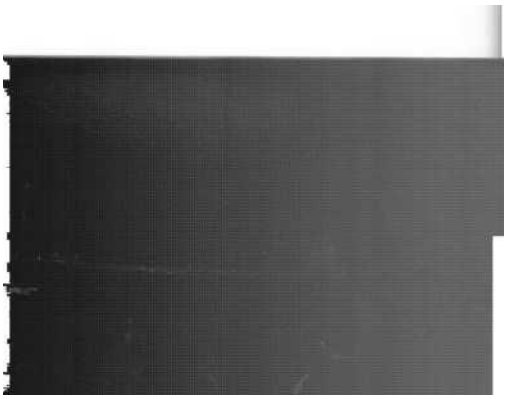
Operatorul `LIKE` permite compararea unui atribut (expresii) cu un literal utilizând o „mască" construită cu ajutorul specificatorilor multipli % şi _. Simbolurile *procent* şi *liniuţăjieJos* (*underscore,* diferită de cratimă) sunt denumite şi *jokeri.* Procentul substituie un şir de lungime variabilă, 0 - n caractere, în timp ce liniuţa substituie un singur caracter.

Care dintre firmele-client sunt societăţi cu răspundere limitată (SRL-uri)?

întrucât, în tabela CLIENŢI, denumirea fiecărui client se termină cu forma sa de societate (SA, SRL etc.), se poate redacta consultarea:

```
SELECT *
FROM CLIENŢI
WHERE DenCl LIKE '%SRL'
```

Rezultatul se prezintă ca în figura 4.12.



CODCL	DENCL	CODFISCAL	ADRESA CODPOST TELEFON
1001	Client 1 SRL	R1001	Tranziţiei, 13 bis 6600
1003	Client 3 SRL	R1003	Prosperităţii, 22 6500 035-222222
1005	Client 5 SRL	R1005	1900 056-111111
1007	Client 7 SRL	R1007	Victoria Capitalismului, 2 1900 056-121212

Figura 4.12. Clienţii - SRL-uri

Care dintre clienţi au numele (fară forma de societate şi spaţiul dinainte) din 8 caractere şi simt societăţi pe acţiuni?

```
SELECT *
FROM CLIENŢI
WHERE DenCl LIKE '__________ SA'
```

Cele şapte liniuţe (nu se observă, dar sunt şapte, parol!) plus spaţiul care le urmează nu au efect vizibil/impresionant pentru baza noastră de date, deoarece în capitolul 1, leneş fiind, am denumit toţi clienţii de o manieră simplistă - Client 1 SRL, Client 2 SA etc. Dacă am avea mai mulţi clienţi (sau măcar un „Client 10 SA"), atunci interogarea ar avea cu totul alt farmec.

Ce persoane au numele conţinând litera Spe a treia poziţie?

```
SELECT *
FROM PERSOANE
WHERE Nume LIKE '    s%'
```

CNP ■-	NUME	PRENUME	ADRESA	SEX	CODPOST	TELACASA	TELBIROU	TELMOBIL	EMAIL
CNP2	Vasile	Ion		B	6600	234567	876543	094222223	Ion@a.ro
CNP6	Vase	Simona	M.Eminescu, 13	F	5725		432109	094222227	

Figura 4.13. Persoane cu litera S pe a treia poziţie a numelui

Rezultatul este vizualizat în figura 4.13. Atenţie! - dacă există persoane al căror nume are litera S majusculă pe a treia poziţie, acestea nu sunt extrase în rezultat. în asemenea situaţii, soluţia de mai jos este ceva mai sigură:

```
SELECT *
FROM PERSOANE
WHERE Nume LIKE ' __ s%' OR Nume LIKE ' ____ S%'[1]
```

In numele căror persoane apare, măcar o dată, litera S (indiferent de poziţie/poziţii)?

```
SELECT *
FROM PERSOANE
WHERE Nume LIKE '%s%' OR Nume LIKE '%S%'
```

CNP	jNUME	PRENUME	ADRESA	SEX	CODPOST	TELACASA	TELBIROU	TELMOBIL	EMAIL
CNP2	Vasile	Ion		B	6600	234567	876543	094222223	Ion@a.ro
CNP6	Vase	Simona	M Eminescu, 13	F	5725		432109	094222227	
CNP8	Bogacs Ildiko		i.V.Viteazu, 67	F	5550	890123	210987	094222229	

Care sunt persoanele ce trebuie felicitate de Sfântul loan?

Firește, am fi tentaţi să redactăm varianta:

```
SELECT *
FROM PERSOANE
WHERE UPPER(Prenume) LIKE '%ION%'
```

Figura 4.14. Persoane al căror nume conţine litera S

Funcţia UPPER face conversia valorilor atributului **Prenume** în majuscule. Rezultatul se vede figura 4.15.

CNP	NUME	PRENUME	ADRESA	SEX	CODPOST	TELACASA	TELBIROU	TELMOBIL	EMAIL
CNP2	Vasile	Ion		B	6600	234567	876543	094222223	Ion@a.ro
CNP4	Lazar	Caraion	M.Eminescu, 42	B	6500	456789		094222225	
CNP5	Iurea	Simion	I.Creanga, 44 bis :B		6500	587890	543210		

Figura 4.15. Tentativă ratată de a extrage „Sfinţii Ioni"

Interogarea nu şi-a atins ţinta, deoarece, pe de o parte, nu au fost extrase persoanele cu prenume ca Ioan, Ioana, Ioanid, iar pe de altă parte, au fost eronat extrase şi persoanele cu prenumele Caraion şi Simion.

Prin urmare, cele trei litere - ION trebuie plasate la începutul prenumelui. La acest şablon se adaugă şi IOAN. Bun, dar dacă pe omul nostru îl cheamă Marius Ion (are două prenume)? Cele două şabloane trebuie să se găsească la începutul fiecărui cuvânt din atributul Prenume. Iar dacă avem în vedere că uneori cele două prenume se despart prin cratimă (Marius-Ion), rezultă o mândreţe de interogare (răspunsul este prezentat în figura 4.16):

```
SELECT *
FROM PERSOANE
WHERE UPPER(Prenume) LIKE 'ION%' OR
   UPPER(Prenume) LIKE 'IOAN%' OR
   UPPER(Prenume) LIKE '% ION%' OR
   UPPER(Prenume) LIKE '% IOAN%'
    OR UPPER(Prenume) LIKE '%-ION%' OR UPPER(Prenume) LIKE '%-IOAN%'
```

CNP	NUME	PRENUME	ADRESA	SEX	CODPOST	TELACASA	TELBIROU	TELMOBiL	EMAIL
CNP2	Vasile	Ion		B	6600	234567	876543	094222223	Ion@a.ro
CNP3	Popovici	Ioana	V.Micle, Bl.I, Sc.B,Ap.2	F	5725	345678		094222224	
CNP7	Popa	Ioanid	I.Ion, BI.H2, Sc.C, Ap.45 B		1900	789012	321098		

Figura 4.16. Persoanele ce trebuie felicitate de Sf. Ioan

Varianta funcţionează în această formă deopotrivă în DB2/Oracle/VFP. Deşi rare, există cazuri când printre caracterele căutate în valorile unui atribut şir de caractere se găseşte chiar unul dintre cele două şabloane, _ sau %.

Dacă, spre exemplu, interesează toţi clienţii care conţin simbolul % în numele lor (s-ar putea ca, la un moment dat, să avem în bază clienţi de genul „Procentul vesel % SRL"). Soluţia este:

```
SELECT *
FROM CLIENŢI
WHERE DenCl LIKE '%\%%' ESCAPE '\'
```

Datorită primului şi ultimului simbol procent, poziţia caracterului căutat (în cazul nostru, chiar %) nu prezintă importanţă: poate fi prima, ultima sau oricare între prima şi ultima. Rezultatul corect este obţinut graţie unui caracter declarat prin clauza ESCAPE. Acesta este backslash-ul, dar poate fi oricare altul. Prin clauza ESCAPE s-a indicat SQL-ului că procentul ce urmează simbolului \ nu este joker, ci are regim de caracter oarecare, ce trebuie căutat în tabelă ca atare.

Operatorul IN

Atunci când se testează dacă valoarea unui atribut este încadrabilă într-o listă de valori dată, în locul folosirii abundente a operatorului OR este mai elegant să se apeleze la serviciile operatorului IN.

Format general:

```
expresiei IN (expresie2, expresie3, ...)
```

Rezultatul evaluării unui predicat ce conţine acest operator va fi *adevărat* dacă valoarea `expresieil` este egală cu (cel puţin) una dintre valorile: `expresie2, expresie3 ...`

Care sunt localităţile din judeţele Iaşi (IS), Vaslui (VS) şi Timiş (TM)? /1/ *4'c. :(~"*

- Fără operatorul `IN`:

```
SELECT Cv                0   -
FROM LOCALITATI
WHERE Jud = 'IS' OR Jud = 'VS' OR Jud = 'TM'
ORDER BY Jud, Loc
```

- Cu operatorul IN:

```
SELECT *
FROM LOCALITATI
WHERE Jud IN ('IS',  'VS',  'TM')
ORDER BY Jud, Loc
```

- Care sunt facturile întocmite pe 1, 3 şi 7 august 2000?

```
SELECT *
FROM FACTURI
WHERE DataFact IN ('01/08/2000',
   '03/08/2000', '07/08/2000')
```

Fireşte, sintaxa trebuie adaptată în funcţie de felul în care fiecare SGBD lucrează cu atribute şi constante de tip dată calendaristică.

4.3. Theta- şi echi-joncţiunea

Dintre tipurile de joncţiune prezentate în capitolul 2, vom insista asupra theta-joncţiunii şi echi-joncţiunii. SQL nu prezintă clauze sau operatori speciali pentru joncţiune, însă aşa cum am văzut, o joncţiune este ..o combinaţie de *produs cartezian* şi *selecţie.* în consecinţă, pentru theta-joncţionarea relaţiilor Rl şI R2'lîm"exeTn'ptn1 nrin"-âlge5rei relaţionale (figura 2.17) se scrie:

```
SELECT *
FROM Rl, R2 WHERE Rl.A >= R2.E
```

iar pentru echi-joncţionarea din exemplul 11 (figura 2.18):

```
SELECT *
FROM Rl, R2 WHERE Rl.A = R2.E
```

Joncţiunea naturală poate fi realizată numai prin specificarea numelor atributelor în clauza `SELECT` a frazei de interogare. în standardul SQL-92 şi în implementările SQL ale multor SGBD-uri se poate folosi o variantă mai elegantă, ţinând seama că tot ce înseamnă theta- şi echi-joncţiune reprezintă, pentru SQL, `INNER JOIN` (joncţiune internă). Prin urmare, cele două soluţii de mai sus pot fi rescrise după cum urmează:

```
SELECT *
FROM Rl INNER JOIN R2 ON Rl.A >= R2.E
```

respectiv

```
SELECT *
FROM Rl INNER JOIN R2 ON Rl.A >= R2.E
```

Surprinzător, cel puţin pentru mine, este că Oracle (Oracle 8) nu are implementată încă această sintaxă, astfel încât, de aici încolo, toate variantele ce utilizează `INNER JOIN` operează în DB2 şi VFP, nu însă şi în Oracle 8.

Reluăm, pentru comparaţie, exemple din algebra relaţională.

Exemplul 13

(Să se obţină, pentru fiecare localitate: codul poştal, denumirea, indicativul şi denumirea judeţului şi regiunea din care face parte).

- Varianta 1 (generală):

```
SELECT CodPost, Loc, LOCALITATI.Jud, Judeţ, Regiune
FROM LOCALITATI, JUDEŢE
WHERE LOCALITATI.Jud'= JUDEŢE.Jud
```

- Varianta 2 (exclusiv Oracle):

```
SELECT CodPost, Loc, LOCALITATI.Jud, Judeţ, Regiune FROM
LOCALITATI INNER JOIN JUDEŢE ON LOCALITATI.Jud = JUDEŢE.Jud
```

Numai atributul `Jud` a fost prefixat de numele tabelei din care provine. Prefixarea este obligatorie atunci când câmpul există în două sau mai multe dintre tabelele enumerate în clauza `FROM`.

Exemplul 14

(Care sunt localităţile din Banat?)

- Varianta 1 (generală):

```
SELECT CodPost, Loc, LOCALITATI.Jud, Judeţ, Regiune FROM
LOCALITATI, JUDEŢE
WHERE LOCALITATI.Jud = JUDEŢE.Jud AND Regiune='Banat'
```

în clauza WHERE, predicatului de selecţie pentru realizarea joncţiunii i s-a adăugat secvenţa de test a regiunii.

- Varianta 2 (exclusiv Oracle):

```
SELECT CodPost, Loc, LOCALITATI.Jud, Judeţ, Regiune FROM
LOCALITATI INNER JOIN JUDEŢE ON LOCALITATI.Jud = JUDEŢE.Jud
WHERE Regiune='Banat'
```

Avantajul celei de-a doua variante ţine de separarea condiţiei ce ţine de regiune, de condiţia legată de joncţiunea propriu-zisă.

Exemplul 15

(In ce zile s-a vândut produsul cu denumirea „ Produs 1 ”?)

- Varianta 1 (generală):

```
SELECT DISTINCT DataFact
FROM PRODUSE, LINIIFACT, FACTURI
WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND
LINIIFACT.Nrfact = FACTURI.NrFact AND
DenPr = 'Produs 1'
```

- Varianta 2 (exclusiv Oracle):

```
SELECT DISTINCT DataFact FROM
PRODUSE
   INNER JOIN LINIIFACT ON PRODUSE.CodPr = LINIIFACT.CodPr INNER JOIN
FACTURI ON LINIIFACT.Nrfact = FACTURI.NrFact WHERE DenPr = 'Produs 1'
```

De notat folosirea clauzei `DISTINCT` pentru eliminarea eventualelor dubluri.

Exemplul 16

(In ce judeţe s-a vândut produsul cu denumirea „Produs 1 ” în perioada 3-5 august 2000?)

- Varianta 1 (generală):

```
SELECT DISTINCT Judeţ
FROM PRODUSE, LINIIFACT, FACTURI, CLIENŢI, LOCALITATI, JUDEŢE
WHERE
   PRODUSE.CodPr = LINIIFACT.CodPr AND LINIIFACT.Nrfact =
   FACTURI.NrFact AND FACTURI.CodCl = CLIENŢI.CodCl AND
   CLIENŢI.CodPost = LOCALITATI.CodPost AND LOCALITATI.Jud =
   JUDEŢE.Jud AND DenPr = 'Produs 1' AND .
      DataFact BETWEEN '03/08/2000' AND '05/08/2000'
```

- Varianta 2 (exclusiv Oracle - atenţie la constantele de tip dată calendaristică!):

```
SELECT DISTINCT Judeţ FROM
PRODUSE
```

```
   INNER JOIN LINIIFACT ON PRODUSE.CodPr = LINIIFACT.CodPr INNER JOIN
   FACTURI ON LINIIFACT.Nrfact = FACTURI.NrFact INNER JOIN CLIENŢI ON
   FACTURI.CodCl = CLIENŢI.CodCl INNER JOIN LOCALITATI ON
      CLIENŢI.CodPost = LOCALITATI.CodPost INNER JOIN JUDEŢE
   ON
      LOCALITATI.Jud = JUDEŢE.Jud WHERE DenPr = 'Produs 1'
AND
   DataFact BETWEEN '03/08/2000' AND '05/08/2000'
```

Şi în acest exemplu este recomandată folosirea clauzei `DISTINCT`.

Exemplul 17

(In ce zile s-au vândut şi produsul cu denumirea „ Produs 1", şi cel cu denumirea „ Produs 2 ”?)

- Soluţie 1 - varianta 1 (nu funcţionează în VFP):

```
SELECT DISTINCT DataFact
FROM PRODUSE, LINIIFACT, FACTURI
WHERE
   PRODUSE.CodPr = LINIIFACT.CodPr AND LINIIFACT.Nrfact =
   FACTURI.NrFact AND DenPr = 'Produs 1'
INTERSECT
SELECT DISTINCT DataFact
FROM PRODUSE, LINIIFACT, FACTURI
WHERE
   PRODUSE.CodPr = LINIIFACT.CodPr AND LINIIFACT.Nrfact =
   FACTURI.NrFact AND DenPr = 'Produs 2'
```

- Soluţie 1 - varianta 2 (SQL-92 şi DB2):

```
SELECT DISTINCT DataFact FROM
PRODUSE
   INNER JOIN LINIIFACT ON PRODUSE.CodPr = LINIIFACT.CodPr INNER JOIN
FACTURI ON LINIIFACT.Nrfact = FACTURI.NrFact WHERE DenPr = 'Produs
1'
INTERSECT
   SELECT DISTINCT DataFact
      FROM PRODUSE INNER JOIN LINIIFACT
           ON PRODUSE.CodPr = LINIIFACT.CodPr INNER
        JOIN FACTURI
           ON LINIIFACT.Nrfact = FACTURI.NrFact WHERE
      DenPr = 'Produs 2'
```

Pentru variantele soluţiei 2 avem nevoie de ceea ce se numeşte joncţiunea tabelei cu ea însăşi, lucru pe care îl vom discuta în paragraful următor.

Exemplul 18

(Ce clienţi au cumpărat şi „ Produs 2 ”, şi „ Produs 3 ”, dar nu au cumpărat „ Produs 5 ”?)

- Soluţia Oracle (şi SQL-92 şi DB2, dacă se înlocuieşte `MINUS` cu `EXCEPT`) :

```
SELECT DISTINCT DenCl
FROM PRODUSE, LINIIFACT, FACTURI, CLIENŢI WHERE
   PRODUSE.CodPr = LINIIFACT.CodPr AND
   LINIIFACT.Nrfact = FACTURI.NrFact AND
   FACTURI.CodCl = CLIENŢI.CodCl AND DenPr =
```

```
    'Produs 2'
  INTERSECT
  SELECT DISTINCT DenCl
  FROM PRODUSE, LINIIFACT, FACTURI, CLIENŢI WHERE
     PRODUSE.CodPr = LINIIFACT.CodPr AND
     LINIIFACT.Nrfact = FACTURI.NrFact AND
     FACTURI.CodCl = CLIENŢI.CodCl AND DenPr =
     'Produs 3'
  MINUS
  SELECT DISTINCT DenCl
  FROM PRODUSE, LINIIFACT, FACTURI, CLIENŢI WHERE
     PRODUSE.CodPr = LINIIFACT.CodPr AND
     LINIIFACT.Nrfact = FACTURI.NrFact AND
     FACTURI.CodCl = CLIENŢI.CodCl AND DenPr =
     'Produs 5'
```

4.4. Sinonime locale şi joncţiunea unei tabele cu ea însăşi

Lucrul cu nume lungi de tabele şi atribute are marele avantaj al lejerităţii la citire şi înţelegerii rapide a logicii de derulare a interogării. în schimb, suficienţi informaticieni nu agreează risipa de caractere (şi, implicit, de timp şi nervi) presupuse de redactările „logoreice". Ambele părţi au dreptate în oarecare măsură (sesizaţi sindromul rabinului!). Astfel încât în frazele `SELECT`, tabelelor li se pot asocia sinonime sau aliasuri mai scurte. Pentru ilustrare, ultima interogare se poate rescrie astfel:

```
SELECT DISTINCT DenCl
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENŢI C WHERE
P.CodPr = LF.CodPr AND LF.Nrfact = F.NrFact AND F.CodCl =
C.CodCl AND DenPr = 'Produs 2'
INTERSECT
SELECT DISTINCT DenCl
FROM PRODUSE P, LINIIFACT   LF, FACTURI F, CLIENŢI    C
WHERE
   P.CodPr = LF.CodPr AND LF.Nrfact = F.NrFact AND F.CodCl
   = C.CodCl AND DenPr = 'Produs 3'
   MINUS
SELECT DISTINCT DenCl
FROM PRODUSE P, LINIIFACT   LF, FACTURI F, CLIENŢI    C
WHERE
   P.CodPr = LF.CodPr AND LF.Nrfact = F.NrFact AND F.CodCl
   = C.CodCl AND DenPr = 'Produs 5'
```

Tabelei PRODUSE i s-a asociat sinonimul P, LINIIFACT LF, FACTURI F, iar pentru CLIENŢI C. Sinonimele prefixează (când este necesar) numele atributelor în clauzele `SELECT` şi `WHERE` (eventual `ORDER BY, GROUP BY`).

Dacă pe calculatorul pe care procesez în acest moment documentul opţiunea `WordCount` funcţionează rezonabil, atunci luăm de bune următoarele cifre: varianta fără sinonime numără 530 de caractere, în timp ce ultima este alcătuită din 431...

Există situaţii în care utilizarea sinonimelor n-are nimic de-a face cu lenea/comoditatea sau depresiile nervoase. în afara interogărilor corelate pe care le vom discuta într-un capitol viitor, o operaţiune în care musai trebuie folosite sinonimele este *jonctiunea unei tabele cu ea însăşi.*

Revenim la *exemplul 19* din algebra relaţională:

Ce facturi au fost emise în aceeaşi zi cu factura 1120?

Este, probabil, cel mai bun exemplu pentru prezentarea subconsultărilor; noi însă ne vom servi acum de acest caz spre a introduce noul tip de joncţiune.

```
SELECT F2.NrFact
FROM FACTURI FI, FACTURI F2
WHERE FI.DataFact = F2.DataFact AND FI.NrFact=1120
```

Joncţiunea tabelei cu ea însăşi înseamnă, de fapt, joncţiunea a două instanţe ale tabelei respective. Rezultatul joncţiunii FI cu F2 este o tabelă „gospodărească", după cum se observă în figura 4.17.

SELECT * FROM FACTURI F1, FACTURI F2 WHERE F1.DataFact-F2.DataFaa

FI.NrFact	FI.DataFact	FI.CodCI	FI.Obs	F2.NtFart	F2.DataFact	F2.CodCl	F2.0bs
1111	1-Aug-00	1001		1111	1-Auq-00	1001	
1112	1-Auq-00	1005	Probleme cu transportul	1111	1-Auq-00	1001	
1113	1-Auq-00	1002		1111	1-Auq-00	1001	
1114	1 -Auq-00	1006		1111	1-Auq-00	1001	
1111	1-Auq-00	1001		1112	1-Auq-00	1005	Probleme cu transportul
1112	1-Auq-00	1005	Probleme cu transportul	1112	1-Auq-00	1005	Probleme cu transportul
1113	1-Auq-00	1002		1112	1-Auq-00	1005	Probleme cu transportul
1114	1-Auq-00	1006		1112	1-Auq-00	1005	Probleme cu transportul
1111	1-Auq-00	1001		1113	1-Auq-00	1002	
1112	1-Auq-00	1005	Probleme cu transportul	1113	1-Auq-00	1002	
1113	1-Auq-00	1002		1113	1-Auq-00	1002	
1114	1-Auq-00	1006		1113	1-Auq-00	1002	
1111	1-Auq-00	1001		1114	1 -Auq-00	1006	
1112	. 1-Auq-00	1005	Probleme cu transportul	1114	1-Auq-00	1006	
1113	1-Auq-00	1002		1114	1-Auq-00	1006	
1114	1-Auq-00	1006		1114	1-Auq-00	1006	
1115	2-Auq-00	1001		1115	2-Auq-QO	1001	
1116	2-Auq-00	1007	Preţul propus iniţial a fost modificat	1115	2-Auq-00	1001	
1115	2-Auq-00	1001		1116	2-Auq-00	1007	Preţul propus iniţial a fost modificat
1116	2-Auq-00	1007	Preţul propus iniţial a fost modificat	1116	2-Auq-OO	1007	Preţul propus iniţial a fost modificat
1117	3-Auq-00	1001		1117	3-Auq-00	1001	
1118	4-Auq-00	1001		1118	4-Auq-00	1001	
1119	/-Auq-00	1003		1119	7-Auq-OO	1003	
1120	7-Auq-OO	1001		1119	7-Auq-OO	1003	
1121	/-Auq-00	1004		1119	7-Auq-OO	1003	
1122	7-Auq-OO	1005		1119	7-Auq-OO	1003	
1119	7-Auq-00	1003		1120	7-Auq-OO	1001	
1120	7-Auq-00	1001		1120	7-Auq-OO	1001	
1121	7-Auq-00	1034		1120	7-Auq-OO	1001	
1122	7-Auq-00	1005		1120	7-Auq-OO	1001	
1119	7-Auq-00	1003		1121	7-Auq-OO	1004	
1120	7-Auq-00	1001		1121	7-Auq-OO	1004	
1121	7-Auq-00	1004		1121	7-Auq-OO	1004	
1122	7-Auq-OO	1005		1121	7-Auq-OO	1004	
1119	7-Auq-00	1003		1122	7-Auq-OO	1005	
1120	7-Auq-OO	1001		1122	7-Auq-OO	1005	
1121	7-Auq-OO	1004		1122	7-Auq-OO	1005	
1122	7-Auq-OO	1005		1122	7-Auq-OO	1005	

Figura 4.17 Joncţiunea a două instanţe ale tabelei FACTURI după DataFact

Asupra acestui rezultat intermediar se aplică predicatul de selecţie suplimentar, `FI. NrFact = 1120,` rezultatul fiind mult mai puţin impresionant - vezi figura `4.18.`

NRFACT	DATAFACT	CODCL	OBS	NRFACT	DATAFACT	CODCL	OBS
1120	07-AUG-00	1001		1119	07-AUG-00	1003	
1120	07-AUG-00	1001		1120	07-AUG-00	1001	
1120	07-AUG-00	1001		1121	07-AUG-00	1004	
1120	07-AUG-00	1001		1122	07-AUG-00	1005	

Figura 4.18. Rezultatul final a! interogării - exemplul 19

O altă variantă, cea care nu funcţionează în Oracle 8, se prezintă astfel: `SELECT`

```
F2.NrFact
FROM FACTURI FI INNER JOIN FACTURI F2 ON
Fl.DataFact = F2.DataFact WHERE
FI.NrFact=1120
```

Ca piatră de încercare, revenim la a doua soluţie formulată în algebra relaţională la ***exemplul 1***7 **:**
în ce zile s-au vândut şi produsul cu denumirea „ Produs 1 ”, şi cel cu denumirea „ Produs 2 ’’?

Joncţionăm o instanţă obţinută prin joncţiunea PRODUSE-LINIIFACT-FACTURI (în care `DenPr = ' Produs 1')` cu o altă instanţă a aceleiaşi combinaţii (în `care DenPr = 'Produs 2').`

- Soluţie 2 - varianta 1 (generală):

```
SELECT DISTINCT FI.DataFact
FROM PRODUSE PI, LINIIFACT LF1, FACTURI FI,
PRODUSE P2, LINIIFACT LF2, FACTURI F2 WHERE
   PI.CodPr = LF1.CodPr AND LF1.NrFact = Fl.NrFact AND
   (PI.DenPr = 'Produs 1') AND P2.CodPr = LF2.CodPr AND
   LF2.NrFact = F2.NrFact AND (P2.DenPr = 'Produs 2') AND
   FI.DataFact=F2.DataFact
```

- Soluţie 2 - varianta 2 (non-Oracle):

```
SELECT DISTINCT Fl.DataFact FROM
PRODUSE PI
   INNER JOIN LINIIFACT LF1 ON PI.CodPr = LF1.CodPr INNER JOIN
   FACTURI FI ON LF1.NrFact = Fl.NrFact INNER JOIN FACTURI F2
   ON FI.DataFact=F2.DataFact INNER JOIN LINIIFACT LF2 ON
   LF2.NrFact = F2.NrFact INNER JOIN PRODUSE P2 ON LF2.CodPr =
   P2.CodPr WHERE
   PI.DenPr = 'Produs 1' AND P2.DenPr = 'Produs 2'
```

Nici nu se putea încheiere mai triumfătoare pentru aşa un paragraf glorios.

4.5. Funcţii-agregat: COUNT, SUM, ÂVG, MIN, MAX

Formatul general al unei fraze SELECT ce conţine funcţii-agregat este:

SELECT `funcţia-predefinităl, ... , funcţia-predefinităN`
FROM `listă-tabele` WHERE
`condiţii.`

în lipsa opţiunii GROUP BY (vezi paragraful următor), dacă în clauza SELECT este prezentă o fimcţie-agregat, rezultatul va conţine o singură linie.

Funcţia COUNT

Funcţia `COUNT` contorizează valorile nenule ale unei coloane sau numărul de linii dintr-un rezultat al unei interogări, altfel spus, în rezultatul unei consultări, `COUNT` numără câte valori diferite de `NULL` are o coloană specificată sau câte linii sunt.

Câţi clienţi are firma?

```
SELECT COUNT (*) AS NrClienti FROM
CLIENŢI
```

Prezenţa asteriscului ca argument al funcţiei `COUNT` are ca efect numărarea liniilor tabelei CLIENŢI. Rezultatul este prezentat în figura 4.19.

NRCLIENTI

Figura 4.19. Câţi clienţi are firma ?

Folosind concatenarea, se poate obţine un rezultat ceva mai elegant. Spre exemplu, în Oracle, prin interogarea următoare se obţine rezultatul din figura 4.20 (la concatenare nu este necesară conversia operanzilor în şiruri de caractere):

```
SELECT 'Firma are '||COUNT (*)!!' clienţi' AS Rezultat FROM
CLIENŢI
```

REZULTAT Firma are 7 clienţi

Figura 4.20. Altă formă de prezentare (în Oracle) a rezultatului funcţiei COUNT Pentru obţinerea

aceluiaşi rezultat în VFP fraza se scrie sub forma:

```
SELECT 'Firma are '+STR(COUNT(*),4)+' clienţi' AS Rezultat FROM
CLIENŢI
```

iar în DB2, în afara operatorului de concatenare `CONCAT`, este necesară conversia rezultatului funcţiei `COUNT`, care este `INTEGER`, în şir de caractere (`CHAR`):

```
SELECT 'Firma are ' CONCAT
   CAST (COUNT (*) AS CHAR(5)) CONCAT ' clienţi' AS Rezultat
FROM CLIENŢI
```

Tabela CLIENŢI are cheie primară atributul `CodCl`, care nu poate avea valori nule; de aceea, la fel de corectă este şi soluţia:

```
SELECT COUNT (CodCl) AS NrClienti FROM
CLIENŢI
```

Câte linii are produsul cartezian al tabelelor FACTURI şi LINIIFACT?

```
SELECT COUNT(*)
FROM FACTURI, LINIIFACT
```

Acum am aflat şi eu: 264.

Pentru câţi clienţi se cunoaşte adresa?

```
SELECT COUNT (Adresa) AS NrClienti
FROM CLIENŢI
```

Rezultatul, 5, putea fi obţinut şi ceva mai complicat, folosind în clauza WHERE operatorul IS NULL, pe care-1 tot amânăm pentru capitolul următor.

Câte facturi au fost emise pe 7 august 2000?

```
SELECT COUNT(NrFact) AS NrFacturi FROM FACTURI
WHERE DataFact = '07/08/2000'
```

Câte facturi au fost emise clienţilor din judeţul Vaslui?

```
SELECT COUNT(NrFact) AS NrFacturi
FROM FACTURI F, CLIENŢI C, LOCALITATI L, JUDEŢE J WHERE F.CodCl = C.CodCl
AND C.CodPost=L.CodPost AND L.Jud = J.Jud AND Judet='Vaslui'
```

Nu ne mai ostenim să redactăm şi forma interogării cu INNER JOIN. Rămâne ca temă pentru acasă, oricare ar fi ea.

în câte localităţi se află clienţii firmei?

Tabela LOCALITĂŢI conţine şi oraşe/comune în care nu se află, momentan, nici un client. De aceea, în locul soluţiei:

```
SELECT COUNT(CodPost) AS NrLocalit FROM LOCALITATI
```

pare mai înţe]epLsIL folosim varianta:....

```
SELECT COUNT(CodPost) AS NrLocalit )
FROM CLIENŢI .
```

Necazul e că rezultatul obţinut, 7, este incorect, deoarece funcţia COUNT numără toate valorile nenule. Există însă o clauză prin care o valoare să se ia în calcul o singură dată: DISTINCT. Rezultatul corect (5) presupune varianta:

```
SELECT COUNT(DISTINCT CodPost) AS NrLocalit FROM CLIENŢI
```

Funcţia SUM

SUM este una dintre cele mai utilizate funcţii în aplicaţiile economice, deoarece datele fmanciar-contabile şi cele ale evidenţei tehnico-operative sunt preponderent cantitative. Probabil că prea multe explicaţii teoretice despre modul în care operează această funcţie sunt inutile, aşa încât vom trece în revistă câteva exemple.

Care este valoarea fără TVA a facturii 1111?

```
SELECT SUM(Cantitate * PretUnit) AS ValFaraTVA FROM LINIIFACT WHERE NrFact =
1111
```

Rezultatul arată ca în figura 4.21.

VALFARATVA
4537500

Figura 4.21. Valoarea fără TVA a facturii 1111 *Care*

este valoarea fără TVA a facturilor emise pe 7 august 2000?

```
SELECT SUM(Cantitate * PretUnit) AS ValFaraTVA_17aug2000
FROM LINIIFACT LF, FACTURI F
WHERE LF.NrFact = F.NrFact AND DataFact = '07/08/2000'
```

VALFARATVAJ7AU02000
11552500

Figura 4.22. Valoarea fără TVA a facturilor emise pe 17 august 2000

Care sunt cele trei valori: fără TVA, TVA şi totală ale facturii 1111?

```
SELECT SUM(Cantitate * PretUnit) AS ValFaraTVA,
   SUM(Cantitate   * PretUnit  * ProcTVA) AS       TVA,
   SUM(Cantitate   * PretUnit  + Cantitate *   PretUnit *
      ProcTVA) AS ValTotala FROM LINIIFACT LF, PRODUSE
P WHERE LF.CodPr = p.CodPr AND NrFact = 1111
```

In condiţiile actuale, în care procentul TVA este unic - 19%, este corectă şi varianta:

```
SELECT SUM(Cantitate * PretUnit) AS ValFaraTVA,
   SUM(Cantitate   * PretUnit  * .19) AS TVA,
   SUM(Cantitate   * PretUnit  + Cantitate *   PretUnit * .19)
      AS ValTotala FROM LINIIFACT WHERE NrFact = 1111
```

Soluţia propusă este una atemporală, altfel spus, a-guvernamentală şi a-relaxare fiscală. O vizualizare mai elegantă a rezultatului poate fi realizată în Oracle utilizând interogarea următoare, rezultatul fiind prezentat în figura 4.23 (analog se pot croi şi variantele în DB2 şi VFP):

```
SELECT 'Pentru factura 1111, valoarea fara TVA este ' i |
   SUM(Cantitate * PretUnit)||
   ', TVA este '||SUM(Cantitate * PretUnit * ProcTVA)||
   ', iar valoarea totala este '||
   SUM(Cantitate * PretUnit + Cantitate *
      PretUnit * ProcTVA) AS Rezultat FROM LINIIFACT,
PRODUSE
WHERE LINIIFACT.CodPr=PRODUSE.CodPr AND NrFact = 1111
```

REZULTAT
Pentru factura 1111, valoarea fara TVA este 4537500, TVA este 862125, iar valoarea totala este 5399625

Dacă se doreşte afişarea pe verticală, sub formă tabelară, a celor trei valori, se pot folosi şabloane (ce diferă de la SGBD la SGBD) împreună cu operatorul reuniune, astfel încât rezultatul va arăta ca în figura 4.24. Am uitat să vă spun că soluţia e valabilă în sintaxa de mai jos tot pentru Oracle 8.

```
SELECT '1' AS X,'Pentru factura 11111' AS TipValoare,
   '' AS Suma FROM dual UNION
   SELECT '2','Valoarea fara TVA',
```

Figura 4.23. Cele trei valori ale facturii 1111 - afişare pe orizontală

```
      TO_CHAR(SUM(Cantitate * PretUnit),'999999999')
   FROM LINIIFACT LF, PRODUSE P WHERE LF.CodPr
= P.CodPr AND NrFact = 1111 UNION SELECT
'3','TVA',
   TO_CHAR(SUM(Cantitate * PretUnit * ProcTVA)999999999') FROM
LINIIFACT LF, PRODUSE P WHERE LF.CodPr = p.CodPr AND NrFact =
1111 UNION
   SELECT '4','Valoarea totala',
   TO_CHAR(SUM(Cantitate*PretUnit*(1+ProcTVA)),'999999999')
FROM LINIIFACT LF, PRODUSE P WHERE LF.CodPr = p.CodPr AND
NrFact = 1111
```

X TIPVALOARE	SUMA
1 Pentru factura 11111	
2 Valoarea fara TVA	4537500
3 TVA	862125
4 Valoarea totala	5399625

Figura 4.24. Cele trei valori ale facturii 1111 - afişare pe verticală

Funcţia `TO_CHAR` converteşte valoarea numerică obţinută prin funcţia `SUM` într-un şir de caractere; prin utilizarea şablonului, valorile vor fi aliniate la dreapta, pentru a fi mai uşor de comparat. Dacă tot am avut ocazia, am folosit şi tabela `DUAL`, o tabelă utilă în Oracle atunci când se doreşte extragerea prin `SELECT` a unei constante. Tabela `DUAL` are o singură linie şi o singură coloană, servindu-ne în acest caz pentru afişarea aşa-zisului antet: *Pentru factura 11111.*

Nu întotdeauna rezultatul reflectă ordinea firească de procesare a liniilor. Este motivul pentru care am introdus în interogare o coloană `(X)`, cu un prim rol decorativ şi un al doilea legat de afişarea liniilor în ordinea care interesează.

La cât se situează cifra vânzărilor pe data de 7 august 2000?

```
SELECT '7 aug. 2000' AS Data,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM LINIIFACT LF, PRODUSE P, FACTURI F
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact
   AND DataFact = '07/08/2000'
```

DATA	VINZARI j
7 aug. 2000	13747475

Figura 4.25. Vânzările zilei de 7 august 2000

Care este valoarea vânzărilor pentru „ Client 1 SRL "?

```
SELECT 'Client 1 SRL' AS Client,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENŢI C
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact AND
F.CodCl = C.CodCl AND DenCl = 'Client 1 SRL'
```

CLIENT	VINZARI
Client 1 SRL	12490240

Figura 4.26. Vânzările către Client 1 SRL

Care este valoarea medie a preţului (inclusiv TVA) la care a fost vândut „Produs 1 "?

Nu, n-am trecut la funcţia `AVG` fără să vă fi anunţat din timp. Soluţia se bazează pe raportul dintre suma valorilor şi cantitatea însumată pentru acest produs.

```
SELECT SUM(Cantitate*PretUnit*(1+ProcTVA)) /
   SUM(Cantitate) AS PretUnitMediu FROM LINIIFACT LF,
PRODUSE P, FACTURI F WHERE LF.CodPr = P.CodPr AND
LF.NrFact = F.NrFact AND DenPr = 'Produs 1'
```

Care este situaţia încasării facturii 1120?

Rezolvarea acestei probleme presupune afişarea, pentru factura 1120, atât a valorii totale, obţinută din tabelele LINIIFACT şi PRODUSE, cât şi a valorii încasate, obţinută din INCASFACT. Prin urmare, pare firesc să încercăm varianta:

```
SELECT '1120' AS NrFact,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat,
   SUM(Transa) AS încasat FROM LINIIFACT LF, PRODUSE P,
FACTURI F, INCASFACT I WHERE LF.CodPr = P.CodPr AND
LF.NrFact = F.NrFact AND F.NrFact=I.NrFact AND F.NrFact
= 1120
```

NRFACT	FACTURAT	INCASAT
1120	1066240	731557

Figura 4.27. Valorile facturată şi încasată ale facturii 1120

Rezultatul din figura 4.27 arată că lucrurile ar fi în regulă. Dar dacă înlocuim numărul facturii 1120 cu 1111, fraza:

```
SELECT '1111' AS NrFact,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat,
   SUM(Transa)         AS        încasat                          .j
```

Y v ° ■L ?

```
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I WHERE
LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact AND
F.NrFact=I.NrFact AND F.NrFact = 1111
```

va genera rezultatul din figura 4.28.

NRFACT	FACTURAT	ÎNCASAT
1111	5399525	16198875

Figura 4.28. Valoarea facturată şi valoarea încasată (greşită !) pentru factura 1111

Valoarea încasată e complet anapoda, cam de trei ori mai mare decât cea facturată. Care e explicaţia? Răspunsul e mai uşor de aflat dacă vizualizăm rezultatul unei interogări „ajutătoare" ce generează tabela pe care se aplică cele două funcţii `SUM` - figura 4.29.

```
SELECT DenPr, Cantitate, PretUnit,
  Cantitate * PretUnit * (1+ProcTVA) AS Facturat,
  Transa                          -
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I WHERE
LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact AND
F.NrFact=I.NrFact AND F.NrFact = 1111
```

DENPR	CANTITATE	PRETUNIT	FACTURAT	TRANSA
Produs 1	50	10000	595000	5399625
Produs 2	75	10500	937125	5399625
Produs 5	500	6500	3867500	5399625

Figura 4.29. Rezultatul joncţiunii „componentelor" facturare şi încasare pentru factura 1111

întâmplarea face ca pentru această factură să existe o singură tranşă de încasare. Tranşa de 5.399.625 lei se repetă pentru fiecare linie a facturii **1111.** Normal că funcţia `SUM` întoarce o valoare triplă faţă de cea reală.
Ce-i de făcut? Să încercăm un artificiu. împărţim suma tranşelor încasate la numărul liniilor facturii:

```
SELECT '1111' AS NrFact,
  SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat,
  SUM(Transa)/COUNT(*) AS încasat FROM LINIIFACT LF,
PRODUSE P, FACTURI F, INCASFACT I WHERE LF.CodPr =
P.CodPr AND LF.NrFact = F.NrFact AND F.NrFact=I.NrFact
AND F.NrFact = 1111
```

Merge! Sau, cel puţin, aşa arată figura 4.30.

NRFACT	[SATURAT	ÎNCASAT
1111	5399625	5399625

Rămâne însă o sursă de crispare: ce se întâmplă cu facturile încasate în mai multe tranşe? Spre exemplu, factura 1117 are două linii şi e încasată în trei tranşe. Dacă utilizăm interogarea „ajutătoare" (penultima), schimbând, bineînţeles, numărul facturii, echivalenta tabelei din figura 4.29 se prezintă ca în figura 4.31.

Figura 4.30. Facturarea şi încasarea - factura 1111

DENPR	CANTITATE	PRETUNIT	FACTURAT	TRANSA
Produs 2	100	10000	1190000	975410
Produs 1	100	9500	1130500	975410
Produs 2	100	10000	1190000	975410
Produs 1	100	9500	1130500	: 975410
Produs 2	100	10000	1190000	369680
Produs 1	100	9500	1130500	369680

Figura 4.31. Valori repetate datorită mai multor tranşe de încasare - factura 1117

Există nu mai puţin de 3 (tranşe de încasare - INCASFACT) * 2 (linii în LINIIFACT) = 6 linii. Pe baza acestei observaţii ochiometrice, putem să încercăm o soluţie finală, împărţind valoarea facturată la numărul tranşelor de încasare şi valoarea încasată la numărul de linii ale facturii respective:

```
SELECT '1117' AS NrFact,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) /
COUNT(DISTINCT I.Codlnc) AS Facturat,
   SUM(Transa)/ COUNT(DISTINCT(LF.Linie)) AS încasat
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact AND
F.NrFact=I.NrFact AND F.NrFact = 1117
```

NRFACT	FACTURAT	INCASAT
1117	2320500	2320500

Figura 4.32. Valorile (corecte) facturată şi încasată - factura 1117

Rezultatul din figura 4.32 ne îndreptăţeşte să sperăm că soluţia e bună. Şi, într-adevăr, este funcţională, dar numai pentru facturile care au *cel puţin o tranşă de încasare!* Pentru cealaltă categorie avem nevoie de cunoştinţe ceva mai avansate (joncţiune externă).

Visual FoxPro nu ne dă, însă, pace. La execuţia acestei variante apare usturătorul `mesaj` de eroare: `SQL: DISTINCT is invalid,` eroare ce are numărul 1819 şi se datorează, după cum scrie „la documentaţie", faptului că se poate utiliza un singur DISTINCT pe nivel. Vom putea remedia situaţia folosind funcţia `MAX`.

Funcţia A VG

Ultimele exemple discutate se doresc o trecere lină la prezentul operator care, după `cum` îi spune şi numele (în engleză), calculează media aritmetică a unei coloane într-o tabelă oarecare, prin divizarea sumei valorilor coloanei respective `la` numărul de valori `nenule ale` acesteia.

Care este valoarea (fară TVA) medie a produselor vândute în factura 1111?

Rezultatul din figura 4.33 se obţine prin interogarea:

```
SELECT 'Val. medie (fara TVA) a prod. din fact. 1111' AS
Explicaţii,AVG(Cantitate * PretUnit) AS ValMedie FROM
LINIIFACT WHERE NrFact = 1111
```

EXPLICAŢII	VALMEDIE
Val. medie (fara TVA) a prod. din fact. 1111	1512500

Figura 4.33. Valoarea medie a produselor din factura 1111

Care este media valorilor (cu TVA) la care a fost vândut„ Produs 1 ”?

```
SELECT 'Val. medie a vinzarilor prod. Produs 1'
   AS Explicaţii,
   ROUND(AVG(Cantitate*PretUnit*(1+ProcTVA)),2) ValTotMedie FROM
LINIIFACT LF, PRODUSE P, FACTURI F WHERE LF.CodPr = P.CodPr AND
LF.NrFact = F.NrFact AND DenPr = 'Produs 1'
```

Adesea, prin aplicarea funcţiei AVG sau prin formularea unor expresii ce conţin rapoarte între atribute/constante, se obţin rezultate cu numeroase zecimale. In cazul de faţă, pentru preîntâmpinarea acestui disconfort vizual şi intelectual, a fost preferată funcţia ROUND, ce rotunjeşte media la două zecimale - vezi figura 4.34.

EXPLICAŢII	VALTOTMEDIE
Val. medie a vinzarilor prod. Produs 1	1128516.67

Figura 4.34. Funcţia AVG combinată cu ROUND

Funcţiile MAX şi MIN

Deosebit de utile în diverse tipuri de analiză, cele două funcţii determină valoarea maximă, respectiv minimă, pentru o coloană (atribut). Se pot folosi şi pentru atribute de tip şir de caractere, caz în care elementul de comparaţie este codul ASCII al caracterelor.

Care este localitatea cu ultima denumire, în ordine alfabetică?

```
SELECT MAX(Loc) AS UltimaLoc FROM LOCALITATI
```

ULTIMALOC
Vaslui

Figura 4.35. Ultima localitate (alfabetic) din baza de date

Care este primul client şi ultimul client (în ordinea numelui) din judeţul Iaşi?

```
SELECT MIN(DenCl) AS Primul_Client,
   MAX(DenCl) AS Ultimul Client
FROM CLIENŢI C, LOCALITATI L, JUDEŢE J WHERE
C.CodPost = L.CodPost AND L.Jud = J.Jud AND Judeţ = 'lasi'
```

PRIMUl_CLIENT	
Client 1 SRL	Client 4

Figura 4.36. Primul şi ultimul client din judeţul laşi

Care este cel mai mare preţ unitar (fară TVA) la care a fost vândut „ Produs 1 ”?

```
SELECT MAX(PretUnit)
FROM LINIIFACT LF, PRODUSE P
WHERE LF.CodPr   = P.CodPr AND    DenPr =    'Produs          1'
```

Din păcate, dacă dorim să aflăm şi în ce factură produsulare preţul unitar maxim, soluţia:

```
SELECT MAX(PretUnit), NrFact FROM LINIIFACT   LF,
PRODUSE P
WHERE LF.CodPr   = P.CodPr AND    DenPr =    'Produs          1'
```

nu funcţionează!

Până la subconsultările din capitolul viitor încercăm o soluţie gen „cârpeală”, con- catenând preţul cu numărul facturii. Varianta următoare este valabilă pentru Oracle:

```
SELECT 'Pret maxim='||MAX(TO_CHAR(LF.PretUnit,'99999999')||
```

```
', apare in fact.'|INrFact) AS Produşi FROM LINIIFACT LF, PRODUSE P
WHERE LF.CodPr = P.CodPr AND DenPr = 'Produs 1'
```

Interogarea este operaţională, cel puţin dacă ne luăm după rezultatul din figura 4.37, aşa că n-are de ce să ne fie jenă de improvizaţie. Rezultatul este incomplet însă, atunci când preţul maxim apare în două sau mai multe facturi, deoarece interogarea de mai sus extrage numai una dintre facturi (cea cu numărul cel mai mare).

```
PR0DUS1
Pret maxim= 10000, apare in fact.1111
```

Figura 4.37. Preţul maxim pentru „Produs 1" şi factura în care apare acest preţ

Care este cel mai mare şi cel mai mic preţ unitar (fără TVA) la care a fost vândut „ Produs 2 "?

Dacă în ultimele exemple am pedalat pe Oracle, iată acum două variante DB2 şi VFP.

- DB2:

```
SELECT MAX( 'Pret maxim=' CONCAT CAST (PretUnit AS CHAR(15)) CONCAT ',
                factura' CONCAT CAST (NrFact AS CHAR(IO)))
                        AS Max_Pret_Produs_2,
 MIN( 'Pret minim=' CONCAT CAST (PretUnit AS CHAR(15)) CONCAT ', factura'
                CONCAT CAST (NrFact AS CHAR(IO)))
            AS Min_Pret_Produs_2 FROM LINIIFACT LF, PRODUSE P
 WHERE LF.CodPr = P.CodPr AND DenPr = 'Produs 2'
```

- VFP:

```
SELECT MAXf'Pret maxim='+STR(PretUnit,12)+',              ;
   factura'+STR(NrFact,8)) AS Max_Pret_Produs_2,          ;
   MIN('Pret minim='+STR(PretUnit,12)+;
         factura'+STR(NrFact,8)) AS Min_Pret_Produs_2 ;
FROM LINIIFACT   LF, PRODUSE P ;
WHERE LF.CodPr   = P.CodPr AND DenPr   =  'Produs    2'
```

Care sunt cele mai mari două preţuri unitare (fără TVA) la care a fost vândut „ Produs 2 "?

După cum vedeţi, o căutăm cu lumânarea. Iată soluţia Oracle:

```
SELECT 'Produs 2:       1||
   MAX('Primul PU: '||TO_CHAR(LF1.PretUnit,'99999999999')|| ', al doilea PU:
   '||TO_CHAR(LF2.PretUnit,'99999999999')I|
   ' - factura primului:'||LF1.NrFact||
       ', factura-al doilea:'||LF2.Nrfact)
   AS "Cele mai     mari doua PretUnit"
FROM LINIIFACT   LF1, LINIIFACT LF2,    PRODUSE     P
WHERE LF1.CodPr = P.CodPr AND DenPr = 'Produs 2'
   AND LF1.CodPr = LF2.CodPr AND
   LF1.PretUnit > LF2.PretUnit
```

Pe acelaşi calapod se redactează soluţiile DB2 şi VFP, a căror sintaxă depinde regulile fiecărui SGBD de conversie între diferite tipuri de date.

Care este situaţia încasării facturii 1120?

A sosit momentul rezolvării problemei VFP care nu permite mai mulţi DISTINCTi pe un nivel de interogare. Soluţia este foarte simplă. în loc să numărăm valorile distincte ale atributului `Linie` în LINIIFACT, determinăm maximul acestui atribut pentru factura luată în considerare. Obţinem astfel varianta VFP, care, firește, funcţionează şi în celelalte două SGBD:

```
SELECT '1120' AS NrFact,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) /
COUNT(DISTINCT I.Codlnc) AS Facturat,
```

```
SUM(Transa) / MAX(LF.Linie) AS încasat FROM LINIIFACT LF,
PRODUSE P, FACTURI F, INCASFACT I WHERE LF.CodPr = P.CodPr
AND LF.NrFact = F.NrFact AND F.NrFact=I.NrFact AND F.NrFact =
1120
```


4.6. Gruparea tuplurilor. GROUP BY şi HAVING

Clauza GROUP BY formează grupe (grupuri) de tupluri ale unei relaţii, pe baza valorilor comune ale unui atribut. în frazele SELECT formulate până în acest paragraf, prin intermediul WHERE au fost selectate *tupluri* ale tabelei. Prin asocierea unei clauze HAVING !a GROUP BY este posibilă selectarea anumitor *grupuri de tupluri* ce îndeplinesc un criteriu, valabil numai la nivel de grup (nu şi la nivel de linie).

Clauza GROUP BY

Rezultatul unei fraze SELECT ce conţine această clauză se obţine prin regruparea tuturor liniilor din tabelele enumerate în FROM, extrăgându-se câte o apariţie pentru fiecare valoare distinctă a coloanei/grupului de coloane. Formatul general este:

```
SELECT coloană 1, coloană 2,                    coloană  m
FROM tabelă

GROUP BY coloană-de-regrupare
```

Care este

valoarea fară TVA a fiecărei facturi emise?

```
SELECT NrFact, SUM(Cantitate*PretUnit) as ValFaraTVA FROM
LINIIFACT GROUP BY NrFact
```

Rezultatul se obţine prin următoarea succesiune de operaţii:

1. Se ordonează liniile tabelei LINIIFACT după atributul de grupare `NrFact`.
2. Se constituie un grup pentru fiecare valoare distinctă a `NrFact`.
3. Se execută funcţia `SUM (Cantitate*PretUnit)` în cadrul fiecărui grup.
4. Se obţine rezultatul al cărui număr de linii coincide cu valorile distincte ale `NrFact`,

Schema simplificată de execuţie a interogării este prezentată în figura 4.38.

Care este valoarea totală a vânzărilor pentru fiecare zi în care s-au emis facturi?

```
SELECT DataFact, SUM(Cantitate * PretUnit * (1+ProcTVÂ))
   AS ValTotala FROM LINIIFACT LF, PRODUSE P, FACTURI F
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact GROUP
BY DataFact
```

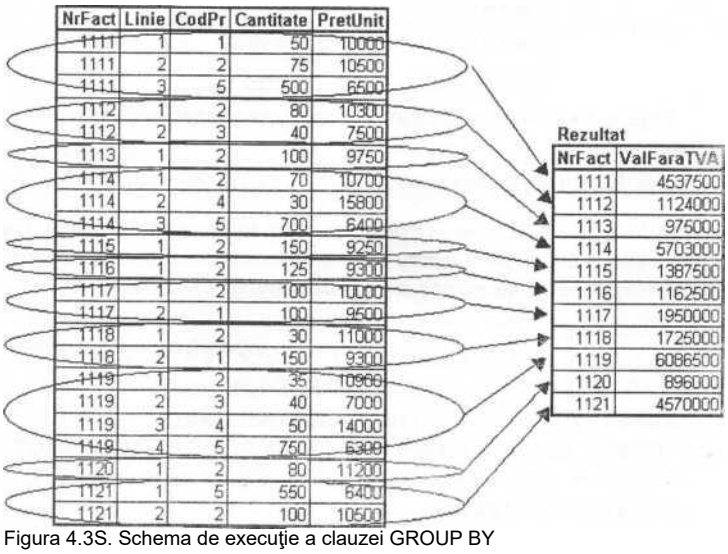


Figura 4.3S. Schema de execuţie a clauzei GROUP BY

Rezultatul este cel din figura 4.39.

DATAFACT	VALTOTALA
01-AUG-00	14684005
2-AUG-00	3034500
3-AUG-00	2320500
04-AUG-00	2052750
07-AUG-00	13747475

Figura 4.39. Vânzările zilnice

Care sunt numărul de facturi şi valoarea vânzărilor pentru fiecare client?

```
SELECT DenCl, COUNT(DISTINCT F.NrFact) AS NrFacturilor,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValTotala FROM
LINIIFACT LF, PRODUSE P, FACTURI F, CLIENŢI C WHERE LF.CodPr =
P.CodPr AND LF.NrFact = F.NrFact AND F.CodCl = C.CodCl GROUP BY
DenCl
```

Analizând rezultatul din figura 4.40 şi comparându-1 cu tabela FACTURI, se observă că al cincilea client ar trebui să aibă două facturi, nu una. Această anomalie aparentă se datorează faptului că factura 1122 nu are nici o linie corespondentă în LINIIFACT, astfel încât această factură „cade" la joncţiune.

DENCL	NRFACTURILOR	VALTOTALA
Clienţi SRL	5	12490240
Client 2 SA	1	1160250
Client 3 SRL	1	7242935
Client 4	1	5438300
Client 5 SRL	1	1337560
Client 6 SA	1	6786570
Client 7 SRL	1	1383375

Figura 4.40. Numărul facturilor şi valoarea vânzărilor pe clienţi

Care este restul de încasat al fiecărei facturi?

Aşa cum reiese din figura 4.41, soluţia următoare oferă un rezultat incomplet. Lipsesc facturile care nu au nici o tranşă de încasare. Pentru rezolvarea cazului, avem nevoie de clauza HAVING, dar şi de

joncţiunea externă, după cum vom vedea în capitolul următor.

```
SELECT F.NrFact,
    SUM(Cantitate * PretUnit * (1+ProcTVA)),
        COUNT(DISTINCT I.Codlnc) AS Facturat,
    SUM(Transa) / MAX(LF.Linie) AS încasat FROM LINIIFACT LF,
PRODUSE P, FACTURI F, INCASFACT I WHERE P.CodPr = LF.CodPr AND
LF.NrFact = F.NrFact AND F.NrFact=I.NrFact GROUP BY F.NrFact
```


NRFACT	FACTURAT	INCASAT
1111	5399625	5399625
1112	1337560	487705
1113	1160250	1160250
1117	2320500	2320500
1118	2052750	2052750
1120	1066240	731557

Figura 4.41. Valorile facturată şi încasată pentru fiecare factură - rezultat incomplet

Care sunt vânzările, cantitativ şi valoric, pentru fiecare produs?

```
SELECT DenPr,
    SUM(Cantitate) AS Cantitativ,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Valoric FROM
LINIIFACT LF, PRODUSE P, FACTURI F WHERE P.CodPr = LF.CodPr
AND LF.NrFact = F.NrFact GROUP BY DenPr
```

Rezultatul - figura 4.42.

DENPR	CANTITATIV	VALORIC
Produs 1	300	3385550
Produs 2	945	11356170
Produs 3	80	690200
Produs 4	80	1397060
Produs 5	2500	19010250

Figura 4.42. Vânzările cantitative şi valorice pe produse

O informaţie esenţială care lipseşte din figura 4.42 este unitatea de măsură, fară de care nu putem î siguri dacă totalul cantitativ se referă la cutii, sticle, pachete, baxuri etc. Din păcate, varianta: SELECT DenPr, UM,

```
    SUM(Cantitate) AS Cantitativ,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Valoric FROM
LINIIFACT LF, PRODUSE P, FACTURI F WHERE P.CodPr = LF.CodPr
AND LF.NrFact = F.NrFact GROUP BY DenPr
```

nu funcţionează, deoarece UM apare separat de atributul de grupare, nefiind inclus în funcţia/ funcţiile care se execută la nivelul grupului. Există însă un remediu simplu: includerea în clauza de grupare şi a atributului UM. Altminteri, gruparea este identică, deoarece DenPr este cheie candidată a tabelei PRODUSE, lucru observabil în figura 4.43.

```
SELECT DenPr, UM,
    SUM(Cantitate) AS Cantitativ,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Valoric FROM
LINIIFACT LF, PRODUSE P, FACTURI F WHERE P.CodPr = LF.CodPr
AND LF.NrFact = F.NrFact GROUP BY DenPr, UM
```

DENPR	UM	CANTITATIV	VALORIC
Produs 1	buc	300	3385550
Produs 2	kci	945 11356170	
Produs 3	Kg	80	690200
Produs 4	l	80	1397060
Produs S	buc	2500	19010250

Figura 4.43. includerea în rezultat **a** unităţii de măsură *Care este*

situaţia vânzărilor pe clienţi şi zile? (_

Interesează valoarea facturilor emise pe clienţi şi, pentru fiecare client, cuantumul zilnic al acestora. Este momentul să folosim o veritabilă grupare după două atribute (spre deosebire de exemplul precendent, când gruparea celor două atribute a fost mai mult de nevoie, decât de voie).

```
SELECT DenCl AS DenumireClient, DataFact AS Data,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
LINIIFACT LF, PRODUSE P, FACTURI F, CLIENŢI C WHERE P.CodPr =
LF.CodPr AND LF.NrFact = F.NrFact AND F.CodCl = C.CodCl GROUP
BY DenCl, DataFact
```

DENUMIRECLIENT	DATA	VINZARI
Client 1 SRL	01-AUG-00	5399625
Client 1 SRL	02-AUG-00	1651125
Client 1 SRL	03-AUG-00	2320500
Client 1 SRL	04-AUG-00	2052750
Client 1 SRL	07-AUG-00	1066240
Client 2 SA	01 -AUG-00	1160250
Client 3 SRL	07-AUG-00	7242935
Client 4	07-AUG-00	5436300
Client 5 SRL	01-AUG-00	1337560
Client 6 SA	01-AUG-00	6786570
Client 7 SRL	02-AUG-00	1383375

Figura 4.44. Vânzările zilnice pentru fiecare client

Care este situaţia vânzărilor fiecărui produs pe fiecare regiune?

```
SELECT DenPr AS Produs, Regiune,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
LINIIFACT LF, PRODUSE P, FACTURI F, CLIENŢI C, LOCALITATI L,
JUDEŢE J WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact AND
F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP
BY DenPr, Regiune
```

PRODUS	REGIUNE	VINZARI
Produs 1	Moldova	3385550
Produs 2	Banat	2363935
Produs 2	Moldova	8992235
Produs 3	Banat	357000
Produs 3	Moldova	333200
Produs 4	Moldova	1397060
Produs 5	Moldova	19010250

Figura 4.45. Vânzările pe regiuni ale fiecărui produs

Să se obţină situaţia vânzărilor pe produse, regiuni şi zile.

Exemplul de faţă este o bună ocazie de a folosi gruparea după trei atribute şi, implicit, o analiză a vânzărilor pe cele axe tradiţionale: sortimentală, teritorială şi calendaristică.

```
SELECT DenPr AS Produs, Regiune, DataFact AS Ziua,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
LINIIFACT LF, PRODUSE P, FACTURI F, CLIENŢI C, LOCALITATI
L, JUDEŢE J WHERE P.CodPr = LF.CodPr AND LF.NrFact =
F.NrFact AND F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND
L.Jud=J.Jud GROUP BY DenPr, Regiune, DataFact
```

Rezultatul e ceva mai impresionant - vezi figura 4.46.

Fără îndoială, interogările de mai sus constituie un ajutor esenţial în analiza vânzărilor oricărei firme (chiar şi de stat). Ceea ce lipseşte este un ingredient important al oricărui raport de acest gen - subtotalul. Vom recurge, aşa cum v-aţi obişnuit, la improvizaţii. Reluăm una dintre problemele anterioare, modificându-i uşor enunţul:

Să se obţină situaţia vânzărilor pe clienţi şi zile, afişându-se câte un subtotal la nivel de client şi un total general.

Ideea improvizaţiei este să „alipim', pentru fiecare client, după vânzările zilnice, câte o linie care să conţină subtotalul. Iată varianta Oracle şi rezultatul din figura 4.47.

PRODUS	REGIUNE	ZIUA	VINZARI
Produs 1	Moldova	01 -AUG-00	595000
Produs 1	Moldova	3- AU	1130500
Produs 1	Moldova	G-00	1660050
Produs 2	Banat	4- A	980560
Produs 2	Banat	02-AUG-00 01 - UG-00 01 -	1383375
Produs 2	Moldova	AUG-00 AUG-00	2988685
Produs 2	Moldova	2- AU	1651125
Produs 2	Moldova	G-00	1190000
Produs 2	Moldova	3- AU	392700
Produs 2	Moldova	G-00	2769725
Produs 3	Banat	4- A	357000
Produs 3	Moldova	UG-00 07-AUG-	333200
Produs 4	Moldova	00 01-AUG-00	564060
Produs 4	Moldova	07-AUG-00 01-	833000
Produs 5	Moldova	AUG-00 07-	9198700
Produs 5	Moldova	AUG-00 01- AUG-00 07- AUG-00	9311550

Figura 4.46. Gruparea vânzărilor pe produse, regiuni şi zile

```
SELECT DenCl AS DenumireClient,
   TO_CHAR(DataFact,'DD-MM-YYYY') AS Data,
   TO_CHAR(SUM(Cantitate * PretUnit * (1+ProcTVA)),
'999999999999') AS Vinzari FROM LINIIFACT LF, PRODUSE P,
FACTURI F, CLIENŢI C WHERE P.CodPr = LF.CodPr AND
LF.NrFact = F.NrFact AND F.CodCl=C.CodCl GROUP BY DenCl,
DataFact UNION
SELECT DenCl II' - Subtotal' , NULL,
   TO_CHAR(SUM(Cantitate * PretUnit * (1+ProcTVA)),
   '999999999999')
FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENŢI C WHERE
P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact AND
F.CodCl=C.CodCl GROUP BY DenCl UNION
SELECT CHR(255)II'TOTAL GENERAL' , NULL,
   TO_CHAR(SUM(Cantitate * PretUnit * (1+ProcTVA)),
   '999999999999')
FROM LINIIFACT LF, PRODUSE P WHERE P.CodPr = LF.CodPr
```

Cu modificările de rigoare, soluţia este asemănătoare şi în DB2. în VFP trebuie ţinut cont că dimensiunea unei coloane de tip şir de caractere depinde în rezultat de numărul de caractere de pe prima linie a rezultatului. Astfel încât pentru denumirea clientului mai trebuie asigurat spaţiu pentru cuvântul ***Subtotal,*** utilizându-se în acest scop funcţia `PADR ()` .

DENUMIRECLIENT	DATA	VINZARI
Client 1 SRL	01-08-2000	5399625
Client 1 SRL	02-08-2000	1651125
Client 1 SRL	03-08-2000	2320500
Client 1 SRL	04-08-2000	2052750
Client 1 SRL	07-08-2000	1066240
Client 1 SRL - Subtotal		12490240
Client 2 SA	01-08-2000	1160250
Client 2 SA - Subtotal		1160250
Client 3 SRL	07-08-2000	7242935
Client 3 SRL - Subtotal	.....	7242935
Client 4	07-08-2000	5438300
Client 4 - Subtotal		5438300
Client 5 SRL	01-08-2000	1337560
Client 5 SRL - Subtotal		1337560
Client 6 SA	01 -08-2000	6786570
Client 6 SA - Subtotal		6786570
Client 7 SRL	02-08-2000	1383375
Client 7 SRL - Subtotal		1383375
yTOTAL GENEI		35839230

Figura 4.47. Vânzările grupate pe clienţi şi zile, cu subtotaluri şi total general - soluţie Oracle

```
SELECT PADR(ALLT(DenCl), 40 ) AS DenumireClient, ;
   DataFact, ;
   SUM ("Cantitate * PretUnit * (1 + ProcTVA)) AS Vinzari ;
FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENŢI C ;
WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact ;
   AND F. CodCI =C. CodC.l ;
GROUP BY DenCl, DataFact ;
UNION ;
   SELECT PADR(ALLT(DenCl)+'-Subtotal',40), ;
      { / / } , ;
      SUM(Cantitate * PretUnit * (1+ProcTVA)) ;
   FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENŢI C ; WHERE
   P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact ;
      AND F.CodCl=C.CodCI ;
   GROUP BY DenCl ;
   UNION ;
SELECT CHR(2 55)+'TOTAL GENERAL' , ;
   {//},;
   SUM(Cantitate * PretUnit * (1+ProcTVA)) ;
FROM LINIIFACT LF, PRODUSE P ;
WHERE P.CodPr = LF.CodPr
```

Rezultatul interogării este afişat în figura 4.48.

-lai *1

	Denumiieclient	Datafact	Vinzari
ti	Client 1 SRL	01/08/2000	5535750.00
	Client 1 SRL	02/08/2000	1692750.00
	Client 1 SRL	03/08/2000	2379000.00
"	Client 1 SRL	04/08/2000	2104500.00
	Client 1 SRL	07/08/2000	1093120.00
	Client 1 SRL-Subtotal	/ /	12805120.00
	Client 2 SA	01/08/2000	1189500.00
	Client 2 SA-Subtotal	/ /	1189500.00;
	Client 3 SRL	07/08/2000	7425530.00!
	Client 3 SRL-Subtotal	/ /	7425530.00!
	Client 4	07/08/2000	5575400.00!
	Client 4-Subtotal	/ /	5575400.00!
	Client 5 SRL	01/08/2000	1371280.00!
	Client 5 SRL-Subtotal	/ /	1371280.001
	Client G SA	01/08/2000	6957660.00;
	Client 6 SA-Subtotal	/ /	6957660.00;
	sPient 7 SRL	02/08/2000	1418250.00!
	! Client 7 SRL-Subtotal	/ /	1418250.00!
	jifOTAL GENERAL	LL.....	36742740.00;

Figura **4.48.** Vânzări pe clienţi şi zile, cu subtotaluri şi total general - soluţie VFP

Care este cea mai mare valoare a unei facturi?

Această interogare este prea pretenţioasă pentru nivelul de SQL la care ne situăm în prezent. Cu toate acestea, Oracle prezintă o facilitate grozavă prin care putem redacta un răspuns: includerea unei funcţii în alta (rezultatul este prezentat în figura 4.49). Din păcate, nici DB2, nici VFP nu „suportă" această facilitate:

```
SELECT MAX(SUM(Cantitate * PretUnit * (1+ProcTVA)))
   AS VaxMaxFact FROM LINIIFACT LF, PRODUSE P WHERE P.CodPr
= LF.CodPr GROUP BY NrFact
```

VAXMAXFACT
7242935

Figura 4.49. Valoarea maximă **a** unei facturi

Clauza HA VING

Cea mai simplă definiţie: clauza HAVING este WHERE-ul ce operează la nivel de grupuri. Dacă WHERE acţionează la nivel de tuplu, selectând acele linii care îndeplinesc o condiţie specificată, HAVING permite specificarea unor condiţii de selecţie care se aplică grupurilor
de linii create prin clauza GROUP BY.

Din rezultat sunt eliminate toate grupurile care nu satisfac condiţia specificată. Formatul
general este:

```
SELECT coloană 1, coloană 2,    .... ,  coloană m
FROM tabelă
GROUP BY coloană-de-regrupare
HAVING caracteristică-de-grup
```

Care sunt zilele în care s-au întocmit cel puţin trei facturi?

```
SELECT DataFact, COUNT(*) AS Nr Facturi FROM FACTURI GROUP BY
DataFact HAVING COUNT(*) >= 3
```

DATAFACT	NR^FACTURI
01 -08-2000	4
07-08-2000	4

Figura 4.50. Zilele în care s-au întocmit trei sau mai multe facturi

Rezultatul este cel din figura 4.50. Logica operaţiunilor este prezentată în figura 4.51.

Pas 1 *Ordonare dupa DataFact*

NrFacs	DataFact	Cod Cl	Obs
1111	01-08-2000	1001	
1112	01-08-2000	1005	Probleme cu transportul
1113	01-08-2000	1002	
1114	01-08-2000	1006	
1115	02-08-2000	1001	
1116	02-08-2000	1007	Preţul propus iniţial a fost modificat
1117	03-08-2000	1001	
1118	04-08-2000	1001	
1119	07-08-2000	1003	
1120	07-08-2000	1001	
1121	07-08-2000	1004	
1122	07-08-2000	1005	

n\\ *Pas 2. Grupare*

DataFact	COUNTn
01-08-2000	4
02-08-2000	2
03-08-2000	1
04-08-2000	1
07-08-2000	4

Pas 3. *Rezultat final*

DataFact	Nr Facturi
01-08-2000	4
07-08-2000	4

Figura 4.51. Logica execuţiei clauzelor GROUP BY şi HAVING

Care sunt clienţii pentru care vânzările depăşesc 5 milioane (lei)?

```
SELECT DenCl AS Client,
SUM(Cantitate * PretUnit * (1+ProcTVA))      AS Vinzari
FROM LINIIFACT LF, PRODUSE P, FACTURI F      WHERE
P.CodPr = LF.CodPr AND LF.NrFact AND         CLIENŢI C
F.CodCl=C.CodCI GROUP BY DenCl               F.NrFact
HAVING SUM(Cantitate * PretUnit *
(1+ProcTVA)) > 5000000
```

CLIENT	VINZARI
Client 1 SRL	12490240
Client 3 SRL	7242935
Client 4	5438300
Client 6 SA	6786570

Figura 4.52. Clienţii cu achiziţii de peste 5 milioane

în ce zile s-au emis mai multe facturi decât pe 2 august 2000?

Cu volumul de cunoştinţe pe care-l avem, putem să redactăm o variantă elegantă, valabilă în SQL-99, DB2 şi Oracle. Soluţia următoare Oracle 8 obţine rezultatul corect, după cum se observă în figura 4.49.

```
SELECT FI.DataFact AS Zii, COUNT(DISTINCT Fl.NrFact)
                                    AS Nr_Facturilorl,
    F2.DataFact AS Zi2, COUNT(DISTINCT F2.NrFact)
                AS Nr__Facturilor2 FROM FACTURI FI, FACTURI F2
WHERE F2.DataFact = TO_DATE('02/08/2000', 'DD/MM/YYYY') GROUP BY
FI.DataFact, F2.DataFact
HAVING COUNT(DISTINCT Fl.NrFact) > COUNT(DISTINCT F2.NrFact)
```

ZI1	NR_FACTURILOR1	Z12	NR_FACTURILOR2
01-AUG-00	4	02-AUG-00	2
07-AUG-00	4	02-AUG-00	2

Figura 4.53. Zilele în **care s-au emis mai multe facturi decât pe 2 august**

Să zăbovim asupra logicii acestei interogări. Mai întâi se efectuează produsul cartezian pentru două instanţe (FI şi F2) ale tabelei FACTURI, eliminându-se pentru F2 liniile în care data este alta decât 2 august 2000. Rezultatul produsului cartezian urmat de selecţie (nu este vorba nicidecum de joncţiune):

```
SELECT *
FROM FACTURI FI, FACTURI F2
WHERE F2.DataFact = TO_DATE('02/08/2000', 'DD/MM/YYYY') ORDER BY
FI.DataFact, F2.DataFact este cel din figura 4.54.
```

NRFACT 1	DATAFACT	ICODCl	oes	NRFACT	(DATAFACT	CODCL	OBS
1111	01-AUG-00	1001		1115	02-AUG-00	001	
1112	01-AUG-00	1005	Probleme cu transportul	1115	02-AUG-00	001	
1113	01-AUG-00	1002		(1115	02-AUG-00	001	
1114	01-AUG-00	1006		1115	02-AUG-00	1001	
1114	01-AUG-00	1006		1116	02-AUG-00	1007	: Preţul propus iniţial a fost modificat
1113	01-AUG-00	1002		1116	02-AUG-00	1007	Preţul propus iniţial a fost modificat
1112	01-AUG-00	1005	Probleme cu transportul	1116	02-AUG-00	1007	Preţul propus iniţial a fost modificat
1111	01-AUG-00	1001		1116	02-AUG-00	1007	Preţul propus iniţial a fost modificat
1115	02-AUG-00	1001		1115	02-AUG-00	1001	
1116	02-AUG-00	1007	Preţul propus iniţial a fost modificat	1116	02-AUG-00	1007	Preţul propus iniţial a fost modificat
1115	02-AUG-00	1001		1116	02-AUG-00	1007	Preţul propus iniţial a fost modificat
1116	02-AUG-00	1007	Preţul propus iniţial a fost modificat	1115	02-AUG-00	1001	
1117	03-AUG-00	1001		1115	02-AUG-00	1001	
1117	03-AUG-00	1001		1116	02-AUG-00	1007	Preţul propus iniţial a fost modificat
1118	04-AUG-00	1001		1115	02-AUG-00	1001	
1118	04-AUG-00	1001		1116	02-AUG-00	1007	Preţul propus iniţial a fost modificat
1119	07-AUG-00	1003		1115	02-AUG-00	1001	
1120	07-AUG-00	1001		1115	02-AUG-00	1001	
1119	07-AUG-00	1003		1116	02-AUG-00	1007	: Preţul propus iniţial a fost modificat
1120	07-AUG-00	1001		1116	02-AUG-00	1007	Preţul propus iniţial a fost modificat
1122	07-AUG-00	1005		1116	02-AUG-00	1007	Preţul propus iniţial a fost modificat
1121	07-AUG-00	1004		1116	02-AUG-00	1007	Preţul propus iniţial a fost modificat
1121	07-AUG-00	1004		1115	02-AUG-00	1001	
1122	07-AUG-00	1005		11115	02-AUG-00	1001	

Figura 4.54. Rezultatul produsului cartezian pentru F2.DataFact='02/08/2000'

în pasul următor se constituie grupuri pentru fiecare combinaţie de valori (FI. DataFact, F2 . DataFact). F2 . DataFact are aceeaşi valoare, 02/08/2000, prin urmare grupurile se constituie în funcţie de Fl.DataFact. Primul grup acoperă primele 8 linii, în care FI. DataFact este 01/08/2000, al doilea 4 linii ş.a.m.d. Pentru ca funcţia COUNT să calculeze corect numărul facturilor dintr-o zi, este necesară clauza DISTINCT.

Care sunt judeţele în care volumul vânzărilor este superior celui al judeţului Neamţ?
Ideea rezolvării este preluată din exemplul anterior. Problema cea mai dificilă ţine de luarea în calcul o singură dată a fiecărei linii dintr-o factură. Cum, la un moment dat, pot apărea în două facturi diferite linii în care cantităţile şi preţurile unitare să fie identice, am introdus, ca artificiu, împărţirea numărului facturii la el însuşi şi a numărului de linie la el însuşi. Varianta următoare funcţionează în Oracle 8:

```
SELECT Jl.Judeţ, J2.Judeţ,
   SUM(DISTINCT (LF1.NrFact/LFl.NrFact) *
      (LF1.Linie/LFl.Linie) *
      LF1.Cantitate * LF1.PretUnit *
      (1 + PI.ProcTVA))
   AS Vinzaril,
   SUM(DISTINCT (LF2.NrFact/LF2.NrFact) *
      (LF2.Linie / LF2.Linie) *
      LF2.Cantitate * LF2.PretUnit *
      (1 + P2.ProcTVA))
   AS Vinzari2 FROM JUDEŢE Jl, LOCALITATI LI,
   CLIENŢI CI, FACTURI FI,
   LINIIFACT LF1, PRODUSE PI,
      JUDEŢE J2, LOCALITATI L2,
      CLIENŢI C2, FACTURI F2,
      LINIIFACT LF2, PRODUSE P2 WHERE Jl.Jud=Ll.Jud AND
LI.CodPost=Cl.CodPost AND CI.CodCl=Fl.CodCl AND
FI.NrFact=LFl.NrFact AND LFl.CodPr=Pl.CodPr AND
Jl.Judet='Neamţ' AND J2.Jud=L2.Jud AND L2.CodPost=C2.CodPost
AND C2.CodCl=F2.CodCl AND F2.NrFact=LF2.NrFact AND
LF2.CodPr=P2.CodPr GROUP BY Jl.Judeţ, J2.Judeţ HAVING
   SUM(DISTINCT (LF1.NrFact/LF1.NrFact)*
      (LF1.Linie/LFl.Linie) *
      LF1.Cantitate * LF1.PretUnit * (1 + PI.ProcTVA))
   <
   SUM(DISTINCT (LF2.NrFact/LF2.NrFact) *
      (LF2.Linie / LF2.Linie) *
      LF2.Cantitate * LF2.PretUnit *
      (1 + P2.ProcTVA))
```

Diviziunea relaţională

Multe situaţii ce reclamă diviziunea relaţională pot fi soluţionate elegant cu ajutorul clauzelor GROUP BY şi HAVING. în exemplul următor este vorba despre o intersecţie „simulată" printr-un mecanism apropiat de diviziune.

în ce zile s-au vândut şi produsul cu denumirea „ Produs 1 ", şi cel cu denumirea „ Produs 2 ''?

```
SELECT DISTINCT DataFact FROM PRODUSE, LINIIFACT, FACTURI
WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND LINIIFACT.Nrfact
```

```
=FACTURI.NrFact AND DenPr IN ('Produs 1',        'Produs 2')
GROUP BY DataFact
HAVING COUNT(DISTINCT LINIIFACT.CodPr) = 2
```

Pentru fiecare grup asociat unei zile de vânzări se numără câte produse, dintre *Produs 1* şi *Produs 2,* au fost facturate. Funcţia COUNT () din clauza HAVING poate „întoarce" maxim valoarea 2, caz în care data respectivă se încadrează în zilele căutate.

Una dintre facilităţile SQL ţine de includerea în predicatul clauzei HAVING nu numai a constantelor şi/sau variabilelor, ci şi a altor consultări (subconsultări). Dar despre această facilitate vom vorbi în capitolul viitor.

Capitolul 5 SQL (CEVA MAI) AVANSAT

Precedentul capitol a avut drept obiectiv explicarea mecanismului de interogare a bazelor de date prin SQL. în cele ce urmează avansăm, treptat, spre ape mai adânci ale limbajului şi vom atinge elemente care se apropie de lumea bună a SQL: valori nule, oncţiuni externe, structuri alternative şi subconsultări.

5.1. Prelucrarea valorilor nule

Valoarea NULL a fost introdusă în primul capitol, la explicarea noţiunilor modelului relaţional, ca posibilitate de reprezentare a informaţiilor... inexistente sau inaplicabile[32]. Raportul Interim 75-02-09 înaintat ANSI X3 (SPARC Study Group 1975) a delimitat 14 tipuri de date incomplete ce ar putea apărea ca rezultate ale unor operaţii sau valori ale atributelor, printre care: depăşiri ale capacităţii de stocare, diviziune la zero, trunchierea şirurilor de caractere, ridicarea lui zero la puterea zero şi alte erori computaţionale, precum şi valori necunoscute.

La popularea bazei de date, unui client nu i se cunoştea codul fiscal, unor clienţi nu li se ştia adresa sau telefonul. Aceste trei atribute aveau, pe una sau mai multe linii, valoarea NULL.

Aşa cum a fost prezentat în capitolul 3, DB2 şi Oracle permit declararea explicită a atributelor ce nu pot avea valori nule, în timp ce în VFP se pun în evidenţă atributele ce pot avea asemenea valori. Prezentăm câteva exemple dedicate operatorului IS NULL.

Pentru care dintre clienţi nu se cunoaşte adresa?

Soluţia cvasigenerală se bazează pe utilizarea operatorului IS NULL, care extrage toate valorile NULL pentru un atribut:

```
SELECT *
FROM CLIENŢI
WHERE Adresa IS NULL
```

CODCL	DENCL	CODFISCAL	ADRESA	CODPOST	TELEFON
1002	Client 2 SA	R1002		6600	032-212121
1005	Client 5 SRL	R1005		1900	056-111111

Figura 5.1. Clienţii fără adresă

Valoarea NULL nu se confundă cu valoarea 0 - pentru atributele numerice - sau cu valoarea ‘ ‘ - spaţiu - pentru atributele de tip şir de caractere. Drept este că înainte de versiunile Visual, FoxPro nu

avea încorporat mecanismul de tratament sistematic al

52. Vezi şi [FotacheOO-2],
valorilor nule (una din regulile SGBD-urilor relaţionale) şi, pentru a identifica clienţii fară adresa introdusă, se folosea o variantă de genul:

```
SELECT * FROM CLIENŢI WHERE Adresa = " "
```

sau

```
SELECT * FROM CLIENŢI WHERE EMPTY(Adresa)
```

Este important de notat că, în vederea identificării valorilor nule, operatorul are forma IS NULL şi nu =NULL. Prin execuţia frazei SQL:

```
SELECT *
FROM CLIENŢI WHERE
Adresa = NULL
```


se va obţine o tabelă cu 0 (zero) linii. Rezultatul evaluării Adresa = NULL va fi întotdeauna FALSE. DB2 chiar afişează, la execuţia aceastei interogări, un convingător mesaj de eroare: SQL0401N The data types of the operands for the operation "=" are not compatible.

Logica operatorilor NOT, AND şi OR este ilustrată în tabelul 5.1.

Tabelul 5.1. Rezultatele utilizării operatorilor NOT, AND şi AND
Aplicarea operatorului NOT unei condiţii

NOT	TRUE	FALSE	UNKNOWN
	FALSE	TRUE	UNKNOWN

Combinarea a două expresii utilizând operatorul AND

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

Combinarea a două expresii utilizând operatorul OR

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE.	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Deşi baza de date prezentată este alcătuită deja dintr-un număr considerabil de tabele şi atribute, introducem încă două tabele cu scop colateral temei „vânzări/încasări" - acela de a gestiona o parte din datele privind drepturile băneşti (salariu negociat şi sporuri) ale angajaţilor firmei. Prima se numeşte PERSONAL2 şi conţine date generale despre angajaţi: marcă; nume şi prenume; data naşterii; compartiment; marca şefului (direct); salariu tarifar. A doua, SPORURI, evidenţiază sporurile lunare primite de fiecare angajat: sporul de vechime (SporVechime), sporul pentru orele lucrate noaptea (SporNoapte), sporuri pentru condiţii deosebite (SporCD) şi sporuri diverse (AlteSpor).

Cu ajutorul valorii NULL se poate face diferenţa între angajaţii pentru care nu s-a calculat valoarea sporului pe luna curentă (şi care vor avea în câmpul corespunzător valoarea NULL) şi cei care nu au dreptul la un asemenea spor, adică valoarea este 0. în continuare este prezentat scriptul de creare a celor două tabele, iar în figurile 5.2 şi 5.3 - conţinuturile acestora.

Listing 5.1. Script DB2/0racle de creare a tabelelor SPORURI şi PERSONAL2

```
DROP TABLE sporuri ;
DROP TABLE personal2 ;
CREATE TABLE personal2 (
   marca INTEGER CONSTRAINT pk_personal2 PRIMARY
   KEY, numepren VARCHAR2(40), datanast DATE,
   compart VARCHAR2(20),
   marcasef INTEGER CONSTRAINT fk_personal2
   REFERENCES personal2(marca), saltarifar INTEGER );
CREATE TABLE sporuri ( an INTEGER, luna INTEGER,
   marca INTEGER REFERENCES persona!2 (marca),
   sporvechime INTEGER,
   spornoapte INTEGER,
   sporcd INTEGER,
   altespor INTEGER,
   PRIMARY KEY (an,luna,marca)
```

Figura 5.2. Conţinutul tabelei PERSONAL2

);

Datele din cele două tabele pot fi interpretate în maniera următoare: firma s-a înfiinţat în aprilie 2000, când avea numai trei angajaţi; La momentul curent (iulie 2000) sunt 10 angajaţi.

Care sunt persoanele şi lunile pentru care nu s-a calculat (nu se cunoaşte) sporul pentru condiţii deosebite?

Prin interogarea:


```
SELECT SPORURI.Marca, NumePren, Compart, An, Luna FROM
PERSONAL2, SPORURI WHERE
PERSONAL2.Marca=SPORURI.Marca AND SporCD IS NULL
```

se obţine situaţia din figura 5.4.

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIFAR
1	ANGAJAT 1	01 -JUL-62	DIRECŢIUNE		6000000
2	ANGAJAT 2	11-OCT-77	FINANCIAR		1 4500000
3	ANGAJAT 3	22-AUG-62	MARKETING		1 4500000
4	ANGAJAT 4		FINANCIAR		2 3800000
5	ANGAJAT 5	30-APR-65	FINANCIAR	2	4200000
6	ANGAJAT 6	Q9-NOV-65	FINANCIAR	5	3500000
7	ANGAJAT 7		FINANCIAR	5	2800000
8	ANGAJAT 8	31 -DEC-60	MARKETING	3	2900000
9	ANGAJAT 9	28-FEB-76	MARKETING	3	4100000
10	ANGAJAT 10	29-JAN-72	RESURSE UMANE	1	3700000

AN LUNA	MARCA	SPORVECHIME	SPORNOAPTE	SPORCD	ALTESPOR
2000 4	1	600000	0	0	320000
2000 4	2	300000	450000	0	170000
2000 4	3	450000	560000	1200000	570000
2000 5	1	600000	0	0	0
2000 5	2	300000	450000	0	170000
2000 5	3	450000	0	0	0
2000 5	10	370000	0	0	860000
2000 6	1	600000	0	0	0
2000 6	2	300000	0	0	150000
2000 6	4	190000	150000	880000	150000
2000 6	5	250000	150000	0	50000
2000 6	10	370000	80000	0	60000
2000 7	1	600000	0		
2000 7	2	300000	0	0	170000
2000 7	3	450000	0	0	0
2000 7	4	190000	150000		150000
2000 7	5	420000	0	0	140000
2000 7	6	350000	575000	0	0
2000 7	7	140000	875000	0	0
2000 7	8	290000	0	1500000	0
2000 7	9	300000	560000	775000	0
2000 7	10	370000	0	0	860000

Figura 5.3. Conţinutul tabelei SPORURI

MARCA	NUMEPREN	COMPART	AN	LUNA
1	ANGAJAT 1	DIRECŢIUNE	2000	7
4	^ o ^ ^	FINANCIAR	2000	7

Figura 5.4. Angajaţii pentru care nu s-au operat sporurile pentru condiţii deosebite

Care sunt angajaţii şi lunile în care aceştia nu au primit spor pentru condiţii deosebite?

Atât soluţia, cât şi rezultatul sunt sensibil diferite - vezi figura 5.5.

```
SELECT SPORURI.Marca, NumePren, Compart, An, Luna FROM
```

```
PERSONAL2, SPORURI WHERE PERSONAL2.Marca=SPORURI.Marca AND
SporCD = 0 ORDER BY An, Luna, NumePren
```

MARCA	NUMEPREN	COMPART	AN	LUNA
1	ANGAJAT 1	DIRECŢIUNE	2000	4
2	ANGAJAT 2	FINANCIAR	2000	4
1	ANGAJAT 1	DIRECŢIUNE	2000	5
10	ANGAJAT 10	RESURSE UMANE	2000	5
2	ANGAJAT 2	FINANCIAR	2000	5
3	ANGAJAT 3	MARKETING	2000	5
1	ANGAJAT 1	DIRECŢIUNE	2000	6
10	ANGAJAT 10	RESURSE UMANE	2000	6
2	ANGAJAT 2	FINANCIAR	2000	6
5	ANGAJAT 5	FINANCIAR	2000	6
10	ANGAJAT 10	RESURSE UMANE	2000	7
2	ANGAJAT 2	FINANCIAR	2000	7
3	ANGAJAT 3	MARKETING	2000	7
5	ANGAJAT 5	FINANCIAR	2000	7
6	ANGAJAT 6	FINANCIAR	2000	7
7	ANGAJAT 7	FINANCIAR	2000	7

Figura 5.5. Angajaţii şi lunile pentru care SporCD este zero (neNULL)

Care dintre angajaţi sunt născuţi înainte de 1 ianuarie 1970 şi care după această dată?

Persoanele născute *înainte* (figura 5.6):

```
SELECT *
FROM PERSONAL2
WHERE DataNast < '01-01-1970'
```

şi *după* (figura 5.7):

```
SELECT *
FROM PERSONAL2
WHERE DataNast >= '01-01-1970'
```

MARCA	NUMEPREN	DATANASTIcOMPART	MARCASEF	SALTARIFAR
1	ANGAJAŢI	01-JUL-62 DIRECŢIUNE		6000000
3	ANGAJAT 3	22-AUG-62 MARKETING		1 4500000
5	ANGAJAT 5	30-APR-65 FINANCIAR		2 4200000
6	ANGAJAT 6	09-NOV-65 FINANCIAR		5 3500000
8	ANGAJAT 8	31 -DEC-60 MARKETING		3 2900000

Figura 5.6. Persoane născute înainte de 1 ianuarie 1970

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIFAR
2	ANGAJAT 2	11-OCT-77	FINANCIAR	1	4500000
9	ANGAJAT 9	28-FEB-76	MARKETING	3	4100000
10	ANGAJAT 10	29-JAN-72	RESURSE UMANE	1	3700000

Se observă un lucru curios: dacă reunim mulţimea angajaţilor născuţi înainte de data fixată cu

mulţimea celor născuţi după această dată nu obţinem relaţia iniţială PERSONAL2 (figura 5.8).

```
SELECT *
FROM PERSONAL2
WHERE DataNast < '01-01-1970'
UNION
SELECT *
FROM PERSONAL2
WHERE DataNast >= '01-01-1970'
```


MARCA	NUMEPREN	DATANAST COMPART	MARCASEF	SALTARIFAR
1	ANGAJAT 1	01-JUL-62 DIRECŢIUNE		6000000
2	ANGAJAT 2	11-OCT-77 FINANCIAR	1	4500000
3	ANGAJAT 3	22-AUG-62 MARKETING	1	4500000
5	ANGAJAT 5	30-APR-6S FINANCIAR	2	4200000
6	ANGAJAT 6	09-NOV-65 FINANCIAR	5	3500000
8	ANGAJAT 8	31-DEC-60 MARKETING	3	2900000
9	ANGAJAT 9	28-FEB-76 MARKETING	3	4100000
10	ANGAJAT 10	29-JAN-72 RESURSE UMANE	1	3700000

Figura 5.8. Reuniunea persoanelor născute înainte de 1 ianuarie 1970 cu persoanele născute după această dată

Din tabela obţinută în figura 5.8. lipsesc angajaţii care nu au precizată data naşterii, altfel spus, persoanele „fără vârstă" (precum maestrul Gică Petrescu). Pentru recompunerea tabelei PERSONAL2, în reuniune mai trebuie adăugate şi liniile pentru care `DataNast IS NULL`:

```
SELECT *
FROM PERSONAL2
WHERE DataNast < '01-01-1970'
UNION
   SELECT *
   FROM PERSONAL2
   WHERE DataNast >= '01-01-1970'
    UNION
 SELECT *
FROM PERSONAL2 WHERE DataNast
IS NULL ' ORDER BY Marca       .
```

Acest exemplu este grăitor în privinţa logicii trivalente a modelului relaţional în ceea ce priveşte valorile nule. în continuare, interesează un alt aspect al NULL-itătilor: modul de evaluare a expresiilor în care unul sau mai mulţi operanzi au valori nule.

Care este totalul sporurilor fiecărui angajat pe luna iulie 2000?

Soluţia pare a fi de forma:

```
SELECT SPORURI.Marca, NumePren, Compart,
   SporVechime + SporNoapte + SporCD + AlteSpor AS
      TotalSporuri
```

KP1–

```
FROM PERSONAL2, SPORURI
WHERE PERSONAL2.Marca=SPORURI.Marca AND An
   = 2000 AND Luna=7
```

Din păcate, rezultatul este incorect - vezi figura 5.9 - deoarece, din prezentarea conţinutului tabelei SPORURI (figura 5.3) reiese că, pe luna iulie 2000, ANGAJAT 1 are calculat spor de vechime (600.000 lei), iar ANGAJAT 4 are, pe aceeaşi lună, şi spor de vechime (190.000 lei), .şi de

noapte (150.000 lei), şi alte sporuri (150.000 lei), iar aceste sporuri nu au fost luate în calcul la însumare.

MARCA	NUMEPREN	COMPART	TOTALSPORURI
1	ANGAJAT 1	DIRECŢIUNE	
2	ANGAJAT 2	FINANCIAR	470000
3	ANGAJAT 3	MARKETING	450000
4	ANGAJAT 4	FINANCIAR	
S	ANGAJAT 5	FINANCIAR	560000
6	ANGAJAT 6	FINANCIAR	925000
7	ANGAJAT 7	FINANCIAR	1015000
8	ANGAJAT 8	MARKETING	1790000
9	ANGAJAT 9	MARKETING	1635000
10	ANGAJAT 10	RESURSE UMANE	1230000

Figura 5.9. Rezultatul unei expresii în care cel puţin un operand este NULL

Explicaţia este simplă: dacă, într-o expresie, unul dintre operanzi este `NULL`, atunci rezultatul evaluării întregii expresii este `NULL`. Fac excepţie funcţiile statistice. Dacă, spre exemplu, vrem să calculăm:

Totalul sporurilor de noapte acordate pentru luna iulie 2000, fraza:

```
SELECT SUM(SporNoapte) AS Total_SporuriNoapte_Luna_7 FROM
SPORURI
WHERE An = 2000 AND Luna=7
```

calculează corect rezultatul: 2.160.000 lei.

Revenim la cazul cu probleme. Pentru a asigura corectitudinea totalului, ar trebui ca în expresie orice valoare nulă să fie considerată zero. Lucru realizabil, deoarece SQL-92 este „prevăzut" cu o funcţie în acest sens – `COALESCE`:

```
SELECT SPORURI.Marca, NumePren, Compart,
   COALESCE(SporVechime,0) + COALESCE (SporNoapte,0) +
COALESCE (SporCD,0) + COALESCE (AlteSpor,0) AS
TotalSporuri FROM PERSONAL2, SPORURI
WHERE PERSONAL2.Marca=SPORURI.Marca AND An
   = 2000 AND Luna=7
```

Sumele obţinute sunt în acest caz cele din figura 5.10.

MARCA	NUMEPREN	COMPART	TOTALSPORURI
1	ANGAJAT 1	DIRECŢIUNE	600000
2	ANGAJAT 2	FINANCIAR	470000
3	ANGAJAT 3	MARKETING	450000
4	ANGAJAT 4	FINANCIAR	490000
5	ANGAJAT 5	FINANCIAR	560000
6	ANGAJAT 6	FINANCIAR	925000
7	ANGAJAT 7	FINANCIAR	1015000
8	ANGAJAT 8	MARKETING	1790000
9	ANGAJAT 9	MARKETING	1635000
10	ANGAJAT 10	RESURSE UMANE	1230000

Figura 5.10. Conversia valorilor nule în zero şi evaluarea corectă a expresiei

în DB2, de obicei, funcţiei COALESCE îi este preferată o alta ce produce rezultate identice - VALUE. Prin urmare, ultima soluţie se poate rescrie astfel:

```
SELECT SPORURI.Marca, NumePren, Compart,
    VALUE (SporVechime,0) + VALUE(SporNoapte,0) +
    VALUE (SporCD,0) + VALUE (AlteSpor,0) AS TotalSporuri FROM
PERSONAL2, SPORURI
WHERE PERSONAL2.Marca=SPORURI.Marca AND An = 2000 AND Luna=7
```

Oracle şi Visual FoxPro pun la dispoziţie o altă funcţie ce acţionează după aceeaşi logică - NVL (de la NullVaLue), astfel încât varianta comună pentru interogarea anterioară este:

```
SELECT SPORURI.Marca, NumePren, Compart,
    NVL(SporVechime,0) + NVL(SporNoapte,0) +
    NVL(SporCD,0) + NVL(AlteSpor,0) AS TotalSporuri FROM
PERSONAL2, SPORURI
WHERE PERSONAL2.Marca=SPORURI.Marca AND
    An = 2000 AND Luna=7
```

Este important de reţinut că funcţiile VALUE, COALESCE, NVL nu se aplică la nivel de expresie, ci fiecărui operand susceptibil de nulitate. De asemenea, un alt element interesant legat de valorile nule ţine de conversia în sens invers, dintr-o valoare oarecare, în NULL.

Să se determine totalul sporurilor de noapte pentru luna iulie 2000, dar, în rezultat, să nu fie luate în calcul valoarea (valorile) 300.000 lei.

Soluţia „clasică" este:

```
SELECT SUM(SporVechime)
FROM SPORURI
WHERE An = 2000 AND Luna=7 AND SporVechime <> 300000
```

O variantă ceva mai elegantă se redactează prin utilizarea funcţiei NULLIF prezentă în SQL-92 care, în interogarea de mai jos, converteşte orice apariţie a valorii 300 000 în NULL:

```
SELECT SUM(NULLIF(SporVechime,300000))
FROM SPORURI
WHERE An = 2000 AND Luna=7
```

în Oracle nu există funcţia NULLIF, dar în locul său se poate folosi REPLACE:

```
SELECT SUM(REPLACE(SporVechime,300000,NULL))
FROM SPORURI
```

```
WHERE An = 2000 AND Luna=7
```

Visual FoxPro nu dispune de nici una dintre funcţiile de mai sus, dar, după cum vom vedea într-un viitor paragraf, conversia (substituirea) valorilor este posibilă prin IF-ul imediat (I IF-ul).

5.2. Joncţiunea externă

Standardul SQL-2 introduce operatorii necesari joncţiunii externe:

- LEFT OUTER JOIN pentru joncţiune externă la stânga,
- RIGHT OUTER JOIN pentru joncţiune externă la dreapta,
- FULL OUTER JOIN pentru joncţiune externă totală (în ambele direcţii).

Dacă ne raportăm la exemplul teoretic din algebra realaţională (capitolul 2, figura 2.24), atunci joncţiunile externe la stânga, la dreapta şi totală dintre relaţiile Rl şi R2 se transcriu în standardul SQL-92 astfel:

Joncţiune externă la stânga

```
SELECT *
FROM Rl LEFT OUTER JOIN R2 ON Rl.C=R2.C
```

Joncţiune externă la dreapta

```
SELECT *
FROM Rl RIGHT OUTER JOIN R2 ON Rl.C=R2.C
```

Joncţiune externă totală

```
SELECT *
FROM Rl FULL OUTER JOIN R2 ON R1.C=R2.C
```

Atât DB2, cât şi Visual FoxPro au implementaţi aceşti operatori; însă, curios, nici până la versiunea 8 Oracle nu a implementat OUTER JOIN-ul. Totuşi, joncţiunea externă este realizabilă, dar ceva mai greoi, astfel:

Joncţiune externă la stânga

```
SELECT *
FROM Rl, R2
WHERE Rl.C = R2.C (+)
```

Joncţiune externă la dreapta

```
SELECT *
FROM Rl, R2
WHERE Rl.C (+) = R2.C
```

Joncţiune externă totală

Nu există nici o posibilitate directă de a joncţiona total două tabele. Unica soluţie este reuniunea rezultatelor joncţiunii la stânga şi la dreapta:

```
SELECT *
FROM R1, R2
WHERE Rl.C (+) = R2.C
UNION
   SELECT *
   FROM Rl, R2
   WHERE Rl.C = R2.C (+)
```

170Cele trei caractere, (+), care simbolizează joncţiunea externă sunt plasate în dreptul tabelei din dreapta LEFT OUTER JOIN-ului şi celei din stânga RIGHT OUTER JOIN-ului[53]. Un element (suplimentar) creator de confuzii îl constituie faptul că simbolul (cele trei caractere) este plasat diferenţiat, fie înaintea semnului =, în cazul joncţiunii externe la dreapta, fie imediat după atributul din dreapta semnului =, în cazul joncţiunii externe la stânga.

Următorul exemplu este desprins tot din algebra relaţională *(exemplul 20): Care sunt localităţile în care nu avem nici un client?*

```
SELECT *
FROM LOCALITATI LEFT OUTER JOIN CLIENŢI ON LOCALITATI.CodPost =
CLIENŢI.CodPost WHERE CLIENŢI.CodPost IS NULL
```

CODPOST	LOC	JUD	CODCL	DENCL	CODFISCAL	ADRESA	CODPOST	TELEFON
5300	Focşani	VN						
5800	Suceava	SV						
6400	Birlad	VS						

Figura 5.11. Localităţile în care nu sunt clienţi

Care este situaţia încasării facturii 1120?

Revenim la un exemplu din capitolul anterior, de la funcţiile SUM şi MAX. Spuneam, la momentul respectiv, că soluţia formulată este valabilă numai pentru facturile cu măcar o tranşă de încasare. Beneficiind de avantajele joncţiunii externe, se poate redacta o soluţie ameliorată care va funcţiona în orice situaţie.

Pentru obţinerea valorii facturate trebuie împărţită suma totală la numărul de tranşe de încasări (repetarea se producea din cauza joncţiunii). Atunci când nu există nici o tranşă, împărţirea trebuie să se facă la 1; astfel încât expresia valorii totale a facturii va fi: SUM (Cantitate * PretUnit * (1 + ProcTVA)) / COUNT (DISTINCT VALUE (I.Codlnc, 1)).

In schimb, totalul tranşelor încasate (0 dacă nu există nici o tranşă) trebuie împărţit, pentru fiecare factură, la numărul de linii din factură. De aici expresia: SUM (VALUE (Transa, 0)) / MAX (LF.Linie).

53. Vezi şi [Fotache00-3].

- Soluţia SQL-92 şi DB2:

```
SELECT '1120' AS NrFact,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) /
COUNT(DISTINCT VALUE( I.CodInc,l)) AS Facturat,
   SUM(VALUE(Transa,0))/ MAX(LF.Linie) AS încasat FROM
LINIIFACT LF
   INNER   JOIN       PRODUSE P    ON     LF.CodPr = P.CodPr
   INNER   JOIN       FACTURI F    ON     LF.NrFact = F.NrFact
   LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact WHERE
F.NrFact = 1120
```

Aşa cum am văzut, DB2 permite folosirea, în locul VALUE, a funcţiei COALESCE.

- Soluţia echivalentă Visual FoxPro:

```
SELECT '1120' AS NrFact, SUM(Cantitate * PretUnit * (1 + ProcTVA)) / ;
   COUNT(DISTINCT NVL(I.Codlnc, 1)) AS Facturat, ;
   SUM(NVL(Transa,0)) / MAX(LF.Linie) AS încasat ;
FROM LINIIFACT LF ;
   INNER   JOIN       PRODUSE P    ON     LF.CodPr = P.CodPr ;
   INNER   JOIN       FACTURI F    ON     LF.NrFact = F.NrFact ;
   LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact ;
WHERE F.NrFact = 1120
```

- Soluţia Oracle 8:

```
SELECT '1120' AS NrFact, SUM(Cantitate * PretUnit *
   (1+ProcTVA)) /COUNT(DISTINCT NVL(I.Codlnc,1))
      AS Facturat,
   SUM(NVL(Transa,0)) / MAX(LF.Linie) AS încasat FROM LINIIFACT
LF, PRODUSE P, FACTURI F, INCASFACT I WHERE LF.CodPr =
P.CodPr AND LF.NrFact = F.NrFact AND F.NrFact=I.NrFact (+) AND
F.NrFact = 1120
```

Care sunt valorile facturate şi încasate ale fiecărei facturi?

Şi acesta este un caz rezolvat incomplet în capitolul anterior (la prezentarea clauzei GROUP BY). Ambele soluţii prezentate în continuare conduc către un rezultat de forma celui din figura 5.12.

NRFACT	FACTURAT	INCASAT
1111	5399625	5399625
1112	1337560	487705
1113	1160250	1160250
1114	6786570	0
1115	1651125	0
1116	1383375	0
1117	2320500	2320500
1118	2052750	2052750
1119	7242935	0
1120	1066240	731557
1121	5438300	0

Figura 5.12. Valorile facturate şi încasate aie fiecărei facturi

- Soluţia SQL-92, DB2 şi VFP (în VFP se înlocuieşte VALUE cu NVL):

```
SELECT F. NrFact,
   SUM(Cantitate 27 PretUnit * (1+ProcTVA)) /
```

27 Soluţia SQL-92/DB2/Visual FoxPro:

```
SELECT PERSONAL2.Marca, NumePren
```

```
                COUNT(DISTINCT VALUE( I.CodInc,l)) AS Facturat,
   SUM(VALUE(Transa,0))/ MAX(LF.Linie) AS încasat FROM
LINIIFACT LF
   INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr INNER JOIN
FACTURI F ON LF.NrFact = F.NrFact LEFT OUTER JOIN INCASFACT I
ON F.NrFact=I.NrFact GROUP BY F.NrFact
```

- Soluţia Oracle 8:

```
SELECT F.NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA)) / COUNT(DISTINCT
   NVL(I.Codlnc,1)) AS Facturat,
   SUM(NVL(Transa,0))     /  MAX(LF.Linie) AS încasat
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I WHERE
LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact AND F.NrFact=I.NrFact (+)
GROUP BY F.NrFact
```

Care au fost sporurile de noapte acordate angajaţilor pe lunile mai şi iunie 2000?

Situaţia obţinută se referă la două luni. Există însă angajaţi care nu au acest spor pe una sau chiar pe ambele luni. Cu interogarea:

```
SELECT AN, Luna, PERSONAL2.Marca, NumePren, SporNoapte FROM
PERSONAL2, SPORURI
WHERE PERSONAL2.Marca=SPORURI.Marca AND
   an=2000 AND Luna IN (5,6)
ORDER BY NumePren, An, Luna
```

rezultatul arată ca în figura 5.13.

AN	LUNA	MARCA	NUMEPREN (SPORNOAPTE	
2000	5 1		ANGAJAT 1	0
2000	6	1	ANGAJAT 1	0
2000	5	10	ANGAJAT 10	0
2000	6	10	ANGAJAT 10 j	80000
2000	5	2	ANGAJAT 2 j	450000
2000	6	2	ANGAJAT 2	0
2000	5	3	ANGAJAT 3	0
2000	6		ANGAJAT 4	150000
2000	6 5		ANGAJAT 5	150000

```
   SI.SporNoapte AS SporNoapte_Mai,
S2.SporNoapte AS SporNoapte_Iunie FROM
PERS0NAL2
```

Figura 5.13. Sporurile de noapte pe mai şi iunie - varianta 1 de afişare Pentru acest exemplu, interesează însă formatul de prezentare din figura 5.14.

	NUMEPRE	SPORNOAPTE	SPORNOAPTEJ
1	ANGAJAT	0	0
10	ANGAJAT	0	80000
2	ANGAJAT	450000	0
3	ANGAJAT	0	
4	ANGAJAT		150000
5	ANGAJAT		150000
6	ANGAJAT		
7	ANGAJAT		
8	ANGAJAT		
9	ANGAJAT		

Figura 5.14. Sporurile de noapte pe mai şi iunie - rezultatul dorit Schimbăm

alura SELECT-ului:

```
SELECT AN, Luna, PERS0NAL2.Marca, NumePren, SporNoapte FROM
PERS0NAL2 LEFT OUTER JOIN SPORURI ON
PERS0NAL2.Marca=SPORURI.Marca AND An=2000 AND Luna=5 ORDER BY
NumePren, An, Luna
```

Se obţine astfel tabela din figura 5.15. Elementul îmbucurător este că în rezultat au fost incluşi toţi angajaţii. Cei care nu erau angajaţi în această perioadă prezintă valori NULL pentru atributele An şi Luna. Pentru afişarea pe coloane separate a sporurilor de noapte pe lunile mai şi iunie (ca în figura 5.14) sunt necesare două joncţiuni externe ale tabelei PERSONAL2 cu două instanţe ale tabelei SPORURI.

AN	LUNA	MARCA	NUMEPREN	SPORNOAPTE
2000	5	1	ANGAJAT 1	0
2000	5	10	ANGAJAT 10	0
2000	5	2	ANGAJAT 2	450000
2000	5	3	ANGAJAT	0
			ANGAJAT	
		5	ANGAJAT	
		6	ANGAJAT 6	
		7	ANGAJAT 7	
		8	ANGAJAT	
		9	(ANGAJAT 9	

Figura 5.15. Sporurile de noapte pe luna mai

```
LEFT OUTER JOIN SPORURI SI ON PERSONAL2.Marca=Sl.Marca AND 2000=Sl.An
AND 5 = SI.Luna
   LEFT OUTER JOIN SPORURI S2 ON PERSONAL2.Marca=S2.Marca AND
2000=S2.An AND 6 = S2.Luna ORDER BY NumePren
```

- Soluţia Oracle:

```
SELECT PERSONAL2.Marca, NumePren,
   TO_CHAR(SI.SporNoapte,'99999999') AS SporNoapte_Mai,
TO_CHAR(S2.SporNoapte,'99999999') AS SporNoapte_Iunie FROM PERSONAL2,
SPORURI SI, SPORURI S2 WHERE PERSONAL2.Marca=Sl.Marca (+) AND
   51.              An(+)=2000       AND Sl.Luna(+)=5
   AND PERSONAL2.Marca=S2.Marca (+) AND
   52.              An(+)=2000       AND
S2.Luna(+)=6 ORDER BY NumePren
```

Elementul-cheie îl constituie prezenţa operatorului joncţiunii externe în dreptul atributelor An şi Luna. Prin aceasta se includ în rezultat şi liniile din tabela PERSONAL2 care nu prezintă corespondenţă după atributul Marca cu tabela SPORURI pentru cele două luni.

Să se obţină sporurile de noapte pentru al doilea trimestru al anului 2000, atât lunar, cât şi cumulat.

Sunt necesare trei instanţe ale tabelei SPORURI, frazele SELECT devenind supraponderale, după cum se va vedea în continuare.

- Soluţia SQL-92/DB2:

```
SELECT PERSONAL2.Marca, NumePren,
   VALUE(SI.SporNoapte,0) AS Spor_Noapte_Aprilie,
   VALUE (S2 . SporNoapte, 0) AS Spor_Noapte__Mai,
   VALUE(S3.SporNoapte,0) AS Spor_Noapte_Iunie,
   VALUE(SI.SporNoapte,0) + VALUE(S2.SporNoapte,0)+
      ALUE(S3.SporNoapte,0) AS Spor_Noapte_Trim_II FROM
PERSONAL2
   LEFT OUTER JOIN SPORURI SI ON PERSONAL2.Marca=Sl.Marca AND
2000=Sl.An AND 4 = SI.Luna LEFT OUTER JOIN SPORURI S2 ON
PERSONAL2.Marca=S2.Marca AND 2000=S2.An AND 5 = S2.Luna LEFT OUTER
JOIN SPORURI S3 ON PERSONAL2.Marca=S3.Marca AND 2000=S3.An AND 6 =
S3.Luna ORDER BY NumePren
```

- Soluţia VFP se obţine din precedenta, înlocuind VALUE cu NVL.
- Soluţia Oracle 8:

```
SELECT PERSONAL2.Marca, NumePren,
   TO_CHAR(NVL(SI.SporNoapte,0),'9999999999')
      AS Spor_Noapte_Aprilie,
   TO_CHAR(NVL(S2.SporNoapte,0),'9999999999')
      AS Spor_Noapte_Mai,
   TO_CHAR(NVL(S3.SporNoapte,0),'9999999999')
         AS Spor_Noapte_Iunie,
      TO_CHAR(NVL(SI.SporNoapte, 0)+NVL(S2.SporNoapte,0) +
         NVL(S3.SporNoapte, 0), '9999999999')
         AS Spor_Noapte_Trim_II FROM PERSONAL2, SPORURI SI,
   SPORURI S2, SPORURI S3 WHERE PERSONAL2.Marca=S1.Marca
   (+) AND
         51.              An(+)=2000 AND Sl.Luna(+)=4 AND
      PERSONAL2.Marca=S2.Marca (+) AND
         52.              An(+)=2    000 AND S2.Luna(+)=5 AND
      PERSONAL2.Marca=S3.Marca (+) AND
         53.              An(+)=2000 AND S3.Luna(+)=6 ORDER
   BY NumePren
```

Cele trei soluţii ar trebui să conducă la un rezultat asemănător celui din figura 5.16.

MARCA	NUMEPREN	SPOR_NOAPTE_APRILIE	SPOR_NOAPTE_MAI	SPOR_NOAPTEJUNIE	SPOR NOAPTE TRIM II
1	ANGAJAT 1	0	0	0	0
10	ANGAJAT 10	0	0	80000	80000
2	ANGAJAT 2	450000	450000	0	900000
3	ANGAJAT 3	560000	0	0	560000
4	ANGAJAT 4	0	0	150000	150000
5	ANGAJAT 5	0	0	150000	150000
6	ANGAJAT 6	0	0	0	0
7	ANGAJAT 7	0	0	0	0
8	ANGAJAT 8	0	0	0	0
9	ANGAJAT 9	0	0	0	0

Figura 5.16. Sporurile de noapte pe trimestrul al II-lea, pe luni şi cumulat

5.3. Structuri alternative: CASE, DECODE, IIF

SQL este un limbaj neprocedural. Cu toate acestea, începând cu standardul 92, SQL prezintă facilitatea codării structurilor alternative prin clauza `CASE`.

Câţi dintre clienţi sunt din Iaşi (codpoştal 6600) şi câţi din afara Iaşiului?

începem cu o versiune „ajutătoare". Pentru a scrie în dreptul fiecărui client dacă e din Iaşi sau din afara Iaşiului, se foloseşte interogarea:

```
SELECT DenCl, CodCl, CodPost,
   CASE CodPost
      WHEN '6600' THEN ' Din Iaşi'
      ELSE 'Din afara Iaşiului'
   END
      AS Poziţionare FROM CLIENŢI
```

Rezultatul arată ca în figura 5.17, soluţia fiind valabilă şi în DB2. Pentru a răspunde exact la întrebare, se poate redacta o variantă SQL-92/DB2, după cum urmează (rezultatul final este prezentat în figura 5.18):

```
SELECT CASE CodPost
         WHEN '6600' THEN 'Din lasi'
      ELSE 'Din afara lasiului'
   END AS Poziţionare,
      COUNT(*) AS NrClienti FROM
CLIENŢI
GROUP BY CASE CodPost WHEN '6600' THEN 'Din lasi'
         ELSE 'Din afara lasiului"'
   END
```

DENCL	CODCL	CODPOST	POZIŢIONARE
Clierri 1 3RI.	1001	6600	Din lasi
Client 2 SA	1002	£000	Din lasi
Client 3 SRL	1003	6500	Din afara lasiului
Client 4	1004	5725	Din afara lasiului
Client 5 SRL	1005	1900	;Din afara lasiului
Client 8 SA	1006	5550	Din afara lasiului
Client 7 SRL	10G7	1900	Din afara lasiului

Figura 5.17. Atribut calculat pe baza unei secvenţe alternative

POZIŢIONARE	NRCLIENTI
Dir. lasi	2
Din afara lasiului	5

Figura 5.18, Numărul clienţilor ieşeni şi ai celor din afara iaşiuiui

Oracle nu avea până la versiunea 8i structura CASE... WHEN... în schimb, putea fi folosită funcţia DECODE, care are o logică similară. DECODE întoarce o valoare în funcţie de conţinutul unui argument:

```
DECODE (atribut, valoare_testatăl, valoarejreturnatăl, valoare_testat'ă2,
      valoare_returnată2,

      valoare_returnatăN)
```

Valoare retunatăN este echivalenta OTHERWISF.-ului dintr-o structură de tip CASE din orice 3GL. Pentru a obţine tabelul din figura 5,17, soluţia Oracle este:

```
SELECT DenCl, CodCI, CodPost,
DECODE (CodPost, ' 6600      '   Din   lasi.',
   'Din afara lasiului')
AS Poziţionare
FROM CLIENŢI
```

iar pentru răspunsul final la problema pusă:

```
SELECT DECODE (CodPost6600 'Din Iasi',
  'Din afara Iaşiului')AS Poziţionare,
  COUNT(*) AS NrClienti FROM CLIENŢI
GROUP BY DECODE (CodPost,'6600 ', 'Din Iasi',
  'Din afara Iaşiului')
```

O dată cu Oracle 8i însă, se poate vorbi de o nouă apropiere de SQL-92, deoarece este operaţională şi varianta:

```
SELECT CASE
     WHEN CodPost = '6600' THEN 'Din Iasi'
     ELSE 'Din afara Iaşiului' END AS Poziţionare,
  COUNT(*) AS NrClienti FROM CLIENŢI GROUP BY CASE
WHEN CodPost = '6600' THEN 'Din Iasi'
     ELSE 'Din afara Iaşiului' END
```

Visual FoxPro pune la dispoziţie în asemenea situaţii IF-ul imediat - `IIF-ul`:

```
SELECT IIF(CodPost='6600 ', PADR('Din Iasi',20), ;
  'Din afara Iaşiului') AS Poziţionare, ;
  COUNT(*) AS NrClienti ;
FROM CLIENŢI ;
GROUP BY Poziţionare
```

Faţă de DB2 şi Oracle, avantajul VFP este că în GROUP BY nu mai trebuie scrisă întreaga expresie - IIF-ul care desemnează coloana după care se face gruparea. De fapt, chiar este interzisă scrierea expresiei, în locul acesteia fiind indicat numele coloanei calculate (Poziţionare) sau numărul coloanei. Soluţia următoare este echivalentă cu ultima consultare:

```
SELECT IIF(CodPost='6600 ', PADR('Din Iasi',20), ;
  'Din afara Iaşiului') AS Poziţionare, ;
  COUNT(*) AS NrClienti ;
FROM CLIENŢI ;
GROUP BY 1
```

Câţi angajaţi au primit, pe luna iulie 2000, spor pentru condiţii deosebite şi câţi nu?

- Soluţia SQL-92/DB2/Oraele 8i:

```
SELECT CASE WHEN SporCD > 0 THEN 'Au spor CD'
ELSE 'Nu au spor CD' END,
   COUNT(*) AS Nr FROM SPORURI
WHERE An=2000 AND Luna=7
GROUP BY CASE WHEN SporCD > 0
THEN 'Au spor CD' ELSE 'Nu au spor CD' END
```

- Soluţia Oracle „tradiţională” are nevoie de un mic truc pentru ca în DECODE să transformăm intervalele în listă. De aceea se scade 1 din NVL(SporCD, 0) şi, dacă rezultatul are semnul +, înseamnă că s-a acordat sporul respectiv, iar în caz contrar că nu s-a acordat:

```
SELECT DECODE(SIGN( NVL(SporCD,0)-1), 1,
   'Au spor CD', 'Nu au spor CD')
   AS Poziţionare,
      COUNT(*) AS NrClienti FROM SPORURI
WHERE An=2000 AND Luna=7
GROUP BY DECODE(SIGN( NVL{SporCD,0)-1),     1,
   'Au spor CD', 'Nu au spor CD')
```

- Soluţia VFP:

```
SELECT IIF(SporCD > 0, 'Au spor CD', 'Nu au spor CD'),       ;
   COUNT(*) AS Nr ;
FROM SPORURI ;
WHERE An=2000 AND Luna=7 ;
GROUP BY 1
```

Scadenţa fiecărei facturi emise este de 20 de zile. Dacă însă data-limită cade într-o sâmbătă sau dtiminică, atunci scadenţa se mută în lunea următoare. Care sunt zilele scadente în aceste condiţii?

Soluţia DB2 se bazează pe funcţia DAYOFWEEK, care întoarce 1 dacă data-argument se referă la duminică, 2 pentru luni... 6 pentru sâmbătă. DAYNAME afişează numele zilei, iar interogarea următoare va genera rezultatul din figura 5.19.

```
SELECT NrFact, DataFact,
   DataFact + 20 DAYS AS Scadental,
   DAYNAME(DataFact + 20 DAYS) AS NumeZil,
   CASE
      WHEN DAYOFWEEK(DataFact + 20 DAYS) = 6
      THEN DataFact + 22 DAYS
      ELSE
```

```
        CASE WHEN DAYOFWEEK(DataFact +20 DAYS) = 1 THEN
        DataFact +21 DAYS ELSE DataFact + 20 DAYS END
   END AS Scadenta,
   DAYNAME (CASE WHEN DAYOFWEEK(DataFact +20 DAYS) = 6 THEN
        DataFact + 22 DAYS ELSE
           CASE WHEN DAYOFWEEK(DataFact +20 DAYS)=1 THEN
           DataFact +21 DAYS ELSE DataFact + 20 DAYS END
        END ) AS Zi_Scadenta FROM
FACTURI
```

NRFACT	DATAFACT	SCADENTA1	NUMEZI1	SCADENTA	ZI SCADENTA
1111	2000-08-01	2000-08-21	Monday	'2000-08-21	Monday
1112	2000-08-01	2000-08-21	Monday	2000-08-21	Monday
1113	2000-08-01	2000-08-21	Monday	2000-08-21	Monday
1114	2000-08-01	2000-08-21	Monday	2000-08-21	Monday
1115	2000-08-02	2000-08-22	Tuesday	2000-08-22	Tuesday
1116	2000-08-02	2000-08-22	Tuesday	2000-08-22	Tuesday
1117	2000-08-03	2000-08-23	Wednesday	2000-08-23	Wednesday
1118	2000-08-04	2000-08-24	Thursday	2000-08-24	Thursday
1119	2000-08-07	2000-08-27	Sunday	2000-08-28	Monday
1120	2000-08-07	2000-08-27	Suriday	2000-08-28	Monday
1121	2000-08-07	2000-08-27	Sunday	2000-08-28	Monday
1122	2000-08-07	2000-08-27	Sunday	2000-08-28	Monday

Figura 5.19. Scadenţa rectificată a încasării facturilor

Soluţia Oracle 8i2 este una ceva mai simplă, un rol decisiv avându-l, pe lângă structura CASE, puternica funcţie TO_CHAR:

```
     SELECT NrFact, TO_CHAR(DataFact, 'YYYY-MM-DD') AS DataFact,
          TO_CHAR(DataFact + 20,'YYYY-MM-DD') AS Scadental,
  TO_CHAR(DataFact + 20,'DAY') AS NumeZil,
         CASE WHEN TO_CHAR(DataFact + 20,'DAY') = 'SATURDAY'
    THEN TO_CHAR(DataFact + 22,'YYYY-MM-DD')
    ELSE
    CASE WHEN TO_CHAR(DataFact + 20,'DAY') = 'SUNDAY'
              THEN TO_CHAR(DataFact + 21,'YYYY-MM-DD')
              ELSE TO_CHAR(DataFact + 20,'YYYY-MM-DD')
    END
  END AS Scadenta,
  TO_CHAR(
       CASE WHEN TO__CHAR (DataFact + 20,'DAY') = 'SATURDAY'
    THEN DataFact + 22 ELSE
              CASE WHEN TO_CHAR(DataFact + 20,'DAY') =
         'SUNDAY'
         THEN DataFact + 21
      ELSE DataFact + 20 END
  END,
  'DAY') AS Zi_Scadenta FROM FACTURI
```

Visual FoxPro prezintă funcţiile CDOW ***(Character Day Of the Week)*** şi DOW ***(Day Of the Week)***, rezultatul din figura 5.19 fiind obţinut după cum urmează:

```
SELECT NrFact AS Factura, DataFact, ;
  DataFact + 20 AS Scadental, ;
  CDOW(DataFact+20) AS NumeZil, ;
 IIF(DOW(DataFact+20)=6, ;
    DataFact+22, ,
    IIF(DOW(DataFact+20)=1, ;
       DataFact+21, ,
       DataFact+20) ) AS Scadenta, ;
 CDOW(IIF(DOW(DataFact+20)= 6,   ;
```

```
     DataFact+22, ;
     IIF(DOW(DataFact+20)=1,    ;
        DataFact+21,  ;
        DataFact+20) )   ) AS  Zi_Scadenta  ;
FROM FACTURI
```

Să se *afişeze în dreptul fiecărei facturi: valoarea facturată, valoarea încasată, diferenţa de încasat, precum şi un text din care să reiasă dacă factura este fără nici o încasare, încasată parţial sau încasată total (şi cele trei valori).*
Practic, se doreşte o situaţie cum este cea din figura 5.20.

NRFACT	FACTURAT	INCASAT	DIFERENŢA	SITUATIUNE
1111	5339625	3399625	0	INCASATA TOTAL
1112	1337560	487705	849855	încasata parţial
1113	118G2S0	1160250	0	ÎNCASATA TOTAL
1114	6786570	0	6786570	Fara nici o incasare
1115	1651125	0	1651125	Fars nici o incasare
1116	1383375	0	1383375	Fara nici o incasare
1117	2320500	2320500	0	INCASATA TOTAL
1118	2052750	2052750	0	INCASATA TOTAL
1119	7242935	0	7242935	Fara nici o incasare
1120	1066240	731557	334683	încasata parţial
1121	5438300	0	5438300	Fara nici o incasare

Figura 6.20, Situaţia încasării facturilor

```
FROM LINIIFACT LF
   INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr INNER JOIN FACTURI F
ON LF.NrFact = F.NrFact LEFT OUTER JOIN INCASFACT I ON F.NrFact=I
NrFact GROUP BY F.NrFact
```

şi una cei puţin la fel de năucitoare în Oracle:

```
SELECT F. NrFact,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) /
      COUNT(DISTINCT     NVL(     I.CodInc,1))     AS     Facturat,
   SUM(NVL(Transa, 0))/ MAX(LF.Linie) AS încasat,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) /
                  COUNT (DISTINCT NVL ( I.Codlnc,1)) -
                     SUM(NVL(Transa,0))/ MAX(LF.Linie)
      AS Diferenţa,
   DECODE (SUM(NVL(Transa,0))/ MAX(LF.Linie),
      0, 'Fara nici o incasare',
      DECODE (
        SIGN(SUM(Cantitate * PretUnit * (1+ProcTVA))
        COUNT(DISTINCT NVL( I.CodInc,1)) - SUM(NVL(Transa,0)) /
        MAX(LF.Linie)),
           0, 'ÎNCASATA TOTAL', 'încasata parţial''';,/ AS
   Situatiune
FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT WHERE LF.CodPr
= P.CodPr AND LF.NrFact = F.NrFact Al F.NrFact=I.NrFact (+)
GROUP BY'F.NrFact
```

Ne-am folosit de acest exemplu (şi) pentru a include o funcţie DECODE în alta.
Din păcate, în VFP, soluţia:

```
SELECT F. NrFact, ;
   SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
```

```
      COUNT(DISTINCT    NVL(    I.Codlnc,1))    AS    Facturat,    ;
   SUM(NVL(Transa,0))/ MAX(LF.Linie) AS încasat, , SUM(Cantitate *
   PretUnit * {1+ProcTVA)) / ;
      COUNT(DISTINCT NVL( I.Codlnc,1)) - ;
                    SUM(NVL(Transa,0))/ MAX(LF,Linie) ;
   AS Diferenţa, ;
   IIF(SUM(NVL(Transa,0))/ MAX(LF,Linie)=0, ;
      'Fara nici o incasare', ;
      IIF(SUM(Cantitate * PretUnit * (1+ProcTVA)) /
        COUNT(DISTINCT NVL( I.CodInc,1)) > ;
        SUM(NVL(Transa,0))/ MAX(LF.Linie), ;
        'încasata parţial', 'ÎNCASATA TOTAL' )) ;
   AS Situatiune ;
FROM LINIIFACT LF ;
   INNER JOIN PRODUSE P ON LF,CodPr = P.CodPr ;
   INNER JOIN FACTURI E1 ON LF.NrFact = F.NrFact ;
   LEFT OLiER JOIN INCASFACT I ON F,NrFact=I NrFact GROUP BY
F.NrFact
```

se loveşte de un obstacol pe care l-am cunoscut în capitolul precedent: operatorul DISTINCT nu poate fi întrebuinţat decât o singură dată pe consultare/subconsultare. O variantă de lucru ar fi secvenţa următoare, dar rezultatul este unul mult mai puţin spectaculos, după cum se observă în figura 5.21.

```
SELECT F. NrFact,    ;
   IIF(SUM(NVL(Transa,0))/ MAX(LF.Linie)=0,      ;
      'Fara nici o incasare', ;
      IIF(SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
      COUNT(DISTINCT NVL( I.CodInc,1)) > ;
      SUM(NVL(Transa,0))/ MAX(LF.Linie),     ;
         'încasata parţial', 'ÎNCASATA TOTAL '     )) ;
   AS Situatiune ;
FROM LINIIFACT LF ;
   INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr ;
   INNER JOIN FACTURI F ON LF.NrFact ='F.NrFact ;
   LEFT  OUTER  JOIN            INCASFACT I ON F.NrFact=I.NrFact ; .
GROUP BY F.NrFact
```

Figura 5.21. Situaţia încasării facturilor - răspuns parţial în VFP

Aceasta nu înseamnă că problema este insolubilă în VFP, ci doar că trebuie să recurgem la o altă variantă, după cum vom vedea cu alt prilej.

- Soluţia fără CASE, DECODE, IIF

Problema poate fi rezolvată şi fară operatorii pentru codarea structurilor alternative, prin reuniunea facturilor fără nici o tranşă încasată cu facturile încasate parţial şi cu facturile încasate total. Din raţiuni de economie de spaţiu, este prezentată în continuare numai varianta DB2 (SQL-92):

```
SELECT F. NrFact,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) /
      COUNT(DISTINCT VALUE( I.Codlnc,1)) AS Facturat,
   SUM(VALUE(Transa,0))/ MAX(LF.Linie) AS încasat,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) /
         COUNT(DISTINCT VALUE( I.CodInc,1)) -
          SUM(VALUE(Transa,0))/ MAX(LF.Linie)
      AS Diferenţa,
      'Fara nici o incasare' AS Situatiune
```

```
FROM LINIIFACT LF
   INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr INNER JOIN
FACTURI F ON LF.NrFact = F.NrFact LEFT OUTER JOIN INCASFACT I
ON F.NrFact=I.NrFact GROUP BY F.NrFact
HAVING SUM(VALUE(Transa,0))/ MAX(LF.Linie) = 0 UNION
   SELECT F. NrFact,
     SUM(Cantitate * PretUnit * (1+ProcTVA)) /
     COUNT(DISTINCT    VALUE(    I.Codlnc,1))    AS    Facturat,
     SUM(VALUE(Transa,0))/     MAX(LF.Linie)     AS     încasat,
     SUM(Cantitate * PretUnit * (1+ProcTVA)) /
       COUNT(DISTINCT VALUE( I.CodInc,1)) - SUM(VALUE(Transa,0))/
                          MAX(LF.Linie),
     'încasata parţial'
   FROM LINIIFACT LF
     INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr INNER JOIN
   FACTURI F ON LF.NrFact = F.NrFact LEFT OUTER JOIN INCASFACT
   I ON F.NrFact=I.NrFact GROUP BY F.NrFact
   HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) /
     COUNT(DISTINCT VALUE( I.Codlnc,1)) >
     SUM(VALUE(Transa,0))/ MAX(LF.Linie)
     AND SUM(VALUE(Transa,0))/ MAX(LF.Linie) O 0 UNION
SELECT F. NrFa-ct,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) /
     COUNT(DISTINCT VALUE( I.Codlnc,!)) AS Facturat,
   SUM(VALUE(Transa,0))/ MAX(LF.Linie) AS încasat,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) /
     COUNT(DISTINCT VALUE( I.Codlnc,1)) -
     SUM(VALUE(Transa,0))/ MAX(LF.Linie),
   'ÎNCASATA TOTAL'
FROM LINIIFACT LF
   INNER JOIN PRODUSE P ON LF.CodPr =, P.CodPr INNER JOIN
FACTURI F ON LF.NrFact = F.NrFact LEFT OUTER JOIN INCASFACT I
ON F.NrFact=I.NrFact GROUP BY F.NrFact
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) /
     COUNT(DISTINCT VALUE( I.Codlnc,1)) =
     SUM(VALUE(Transa,0))/ MAX(LF.Linie)
```

5,4, Subconsultări **Operatorul** IN

Una din cele mai importante facilităţi ale SQL constă în includerea unei consultări în alta, pe două sau mai multe niveluri - altfel spus, utilizarea *subconsultărilor.* Prin subconsultări se obţin tabele temporare intermediare folosite ca „argumente" ale frazelor SELECT superioare.

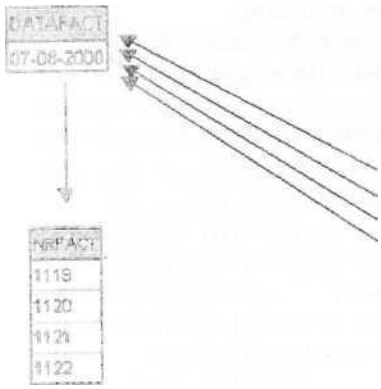
Operatorul cel mai utilizat în materie de subconsultări este IN, pe care l-am întâlnit deja în capitolul precedent într-o cu totul altă ipostază - testarea încadrării valorii unui atribut într-o listă de constante. Pentru, cele ce urmează, domeniul de testare este alcătuit din liniile unei tabele obţinute printr-o (sub)consultare.

Revenim (pentru a doua oară) la *exemplul 19* din algebra relaţională: *Ce facturi au fost emise în aceeaşi zi cu factura 1120?*

In capitolul anterior a fost formulată o soluţie bazată pe joncţiunea a două instanţe ale tabelei FACTURI. Iată şi o soluţie mai simplă, bazată pe subconsultări:

```
SELECT NrFact FROM FACTURI WHERE DataFact IN
{SELECT DataFact FROM FACTURI WHERE
NrFact=1120)
```

Exectţla acestei interogări se derulează în doi timp. Mai întâi se execută subconsultarea SELECT "oataFact FROM FACTURI WHERE NrFact=1120, obţinându-se o tabelă intermediară cu o singură linie (pe care apare 17-08-2000) şi o singură coloană (OataFact), ca în figura 5.22. în al doilea pas sunt selectate liniile tabelei FACTURI care îndeplinesc condiţia Da ta Fact = '17-08-2000'.



FACTOR*

KiFact	DsiaFact	**CodCl**	Obs
/1111	01/08/2000	1001	NULL
1112	01/08/2000	1005	Probleme cu transportul
1113	01/08/2000	1002	NULL
1114	01/08/2000	1006	NULL
1115	02/08/2000	1001	NULL
1118	02/08/2000	1007	Preţul propus iniţial a fost modificat
1117	03/08/2000	1001	NULL
1118	04/08/2000	1001	NULL
1119	07/08/2000	1003	NULL
1120	07/08/2000	1001	NULL
1121	07/08/2000	1004	NULL
1122	07/08/2000	1005	NULL

Figura **5.22.** Schema execuţiei frazei SELECT ce prezintă o subconsultare

în rezultat a fost inclusă şi factura de referinţă - 1120. Dacă se doreşte excluderea acesteia, fraza SELECT se modifică astfel:

```
                SELECT NrFact FROM FACTURI WHERE DataFact IN
    (SELECT DataFact
    FROM FACTURI
    WHERE
    NrFact=1120)
AND NrFact O 1120
```

Ce facturi au fost emise în alte zile decât factura 1120?

Acest exemplu necesită folosirea operatorului de negaţie:

```
SELECT NrFact FROM FACTURI WHERE DataFact NOT IN (SELECT
DataFact FROM FACTURI WHERE NrFact=1120)
```

Care simt clienţii cărora li s-au trimis facturi în aceeaşi zi în care a fost întocmită factura 1120?

```
SELECT DenCl FROM CLIENŢI WHERE CodCI IN (SELECT CodCI FROM
FACTURI WHERE DataFact IN
        (SELECT DataFact
        FROM FACTURI WHERE
        NrFact=1120))
```

Rezultatul prezentat în figura 5.23 se obţine folosind trei niveluri de interogare (fraza principală o subconsultare şi osub-subconsultare).

DENCL -
Client 1
SRL Client
3 SRL
Client 4
Client 5
SRL

Figura 5.23. Clienţii pentru care există facturi emise în aceeaşi zi ca 1120

Dacă în DB2 şi Oracle această ultimă interogare este executată fară probleme, în VFP ..facem cunoştinţă" cu una dintre cele mai serioare limitări SQL - maximum două niveluri de consultare (fraza principală şi o interogare subordonată). Mesajul de eroare are numărul 1842: SQL; Subquery nesting is too deep. _

Astfel, pentru a răspunde la întrebare, soluţia de interogare trebuie reformulată:

```
SELECT DenCl ;
FROM CLIENŢI INNER JOIN FACTURI ;
    ON CLIENŢI.CodCl=FACTURI.CodCI ;
WHERE DataFact IN ;
        (SELECT DataFact ;
        FROM FACTURI ;
        WHERE NrFact=1120)
```

în ce judeţe s-a vândut produsul „ Produs 2 ”?

Am ales acest exemplu pentru a „vântura”, prin subconsultări, cât mai multe tabele ale bazei (rezultatul - vezi figura 5.24):

```
SELECT Judeţ FROM JUDEŢE WHERE Jud IN (SELECT Jud FROM
LOCALITATI WHERE CodPost IN (SELECT CodPost FROM CLIENŢI WHERE
CodCI IN (SELECT CodCI FROM FACTURI WHERE NrFact IN (SELECT NrFact
FROM LINIIFACT WHERE CodPr IN (SELECT CodPr FROM PRODUSE
        WHERE DenPr = 'Produs 2' )      ) ) ) )
```

JUDEŢ

Iasi

Neamţ

Timiş

Vaslui

Figura 5.24. Judeţele în care s-a vândut „Produs 2”

Revenim la tabela PERSONAL2 din figura 5.2: *Câţi subordonaţi direcţi are ANGAJAT2?*

La această problemă (la care răspunsul este 2) formulăm, pentru comparaţie, două soluţii. Soluţia bazată pe joncţiune este:

```
SELECT COUNT(*) AS NrSubordonati FROM PERSONAL2 SUBORDONAŢI,
PERSONAL2 SEFI WHERE SUBORDONAŢI.MarcaSef=SEFI.Marca AND
SEFI.NumePren='ANGAJAT 2'
```

A doua soluţie utilizează o subconsultare:

```
SELECT COUNT(Marca) AS NrSubordonati
FROM PERSONAL2 WHERE MarcaSef IN
(SELECT Marca FROM PERS0NAL2 WHERE
NumePren='ANGAJAT 2')
```

în capitolul 2 am cheltuit destul de puţin timp şi spaţiu pentru un tip de joncţiune nu prea folosit *semijoncţiunea,* prin care se extrag numai liniile dintr-o tabelă ce au corespondent (ca valoare a atributului de legătură) în a doua tabelă. Cu operatorul IN se pot formula lejer soluţii ce corespund semijoncţiunii. Astfel, o tabelă precum cea din figura 2.25 se obţine prin:

```
SELECT *
FROM Rl WHERE C IN (SELECT C FROM R2)
```

Completam discuţia şi cu exemplul „practic” luat pentru ilustrarea semijoncţiunii (exemplul 21 din capitolul 2): *Care sunt localităţile (codul poştal, denumirea şi indicativul judeţului) în care exista macar un client?*

```
SELECT *
FROM LOCALITATI WHERE CodPost IN
(SELECT CodPost FROM CLIENŢI)
```

CODPOST	LOC	JUD
1900	Timişoara	TM
5550	Roman	NT
5725	Paşcani	IS
6500	Vaslui	VS
6600	Iasi	IS

Figura 5.25. Localităţi în care există măcar un client

Tot prin subconsultări putem realiza intersecţia şi diferenţa relaţională. Raportându-ne la exemplul teoretic din capitolul 2 (figura 2.4), *intersecţia* celor două relaţii, Rl şi R2, se poate obţine şi astfel:

```
SELECT *
FROM Rl
WHERE (A,B,C) IN
    (SELECT C,D,E
    FROM R2)
```

Varianta funcţionează în DB2 şi Oracle, nu însă şi în VFP, deoarece într-o subconsultare nu pot fi testate simultan trei valori (ale atributelor A, B şi C), ci numai una. Atfel încât, pentru a-i înşela vigilenţa, vom concatena cele trei atribute, dând senzaţia că avem de-a face cu unul singur-

```
SELECT * ;
FROM R1 ;
WHERE STR(A,4)+B+STR(C,4) IN ;
      (SELECT STR(C,4)+D+STR(E,4)      ;
      FROM R2)
```

Un alt exemplu de intersecţie (exemplul 17 - algebra relaţională):
în ce zile s-au vândut şi produsul cu denumirea „ Produs 1 ”, şi cel cu denumirea „ Produs 2 ?

```
SELECT DISTINCT DataFact FROM PRODUSE, LINIIFACT,. FACTURI
WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND LINIIFACT.Nrfact =
FACTURI.NrFact AND DenPr = 'Produs 1'
   AMD DataFact IN
      (SELECT DISTINCT DataFact FROM PRODUSE,
      LINIIFACT, FACTURI WHERE PRODUSE.CodPr =
      LINIIFACT.CodPr AND LINIIFACT.Nrfact =
      FACTURI.NrFact AND DenPr = 'Produs 2')
```

Cât priveşte *diferenţa* relaţională, cheia rezolvării este NOT IN. Diferenţa relaţiilor R1 şi R2, prezentată în figura 2.5, poate fi calculată în SQL astfel:

```
SELECT *
FROM R1
WHERE (A,B,C) NOT IN {SELECT C,D,E FROM R2)
```

Firește, în VFP vom aplica „trucul” concatenării, ca şi la intersecţie. Apelăm iarăşi la un exemplu din algebra relaţională (exemplul 18):
Ce clienţi au cumpărat şi „ Produs 2 ”, şi „ Produs 3 ", dar nu au cumpărat „ Produs 5 ”?
Soluţia comună SQL-92/DB2/Oracle/Visual FoxPro este:

```
SELECT DISTINCT DenCl
FROM PRODUSE, LINIIFACT, FACTURI, CLIENŢI
WHERE
      PRODUSE.CodPr = LINIIFACT.CodPr AND
      LINIIFACT.Nrfact = FACTURI.NrFact AND
      FACTURI.CodCl = CLIENŢI.CodCl AND DenPr
      = 'Produs 2'
      AND FACTURI.CodCl IN
          (SELECT CodCl
          FROM PRODUSE, LINIIFACT, FACTURI WHERE
          PRODUSE.CodPr = LINIIFACT.CodPr AND
          LINIIFACT.Nrfact = FACTURI.NrFact AND DenPr =
          'Produs 3")
    AND FACTURI.CodCl NOT IN
          (SELECT CodCl
```

mm

```
          FROM PRODUSE, LINIIFACT, FACTURI WHERE PRODUSE.CodPr =
          LINIIFACT.CodPr AND LINIIFACT.Nrfact = FACTURI„NrFact AND DenPr =
          'Produs 5')
```

Până la apariţia operatorilor LEFT OUTER JOIN, RIGHT OUTER JOIN şi FULL OUTER JOIN, în multe SGBD-uri *joncţiunea externă* era realizată prin reuniunea liniilor obţinute din echi-joncţiune cu liniile unei tabele (completate cu zerouri/spaţii pentru atributele
celeilalte tabele) ce nu au corespondent în cealaltă.
Cum joncţiunea externă a fost definită în capitolul 2, revenim la operaţiunea ilustrată în figura 2.24. Joncţiunea externă la stânga a relaţiilor Rl şi R2 prin atributul C ar putea fi realizată şi astfel:

```
SELECT *
FROM Rl, R2 WHERE Rl.C = R2.C UNION
   SELECT A,B,C, 0, *          0
   FROM Rl
   WHERE C NOT IN (SELECT C FROM
         R2)
```

Varianta de mai sus funcţionează în toate cele trei SGBD-uri. La drept vorbind, soluţia pe care am încercat-o prima dată şi care ar fi corespuns pe deplin joncţiunii externe era:

```
SELECT *
FROM Rl, R2 WHERE Rl.C = R2.C UNION
                '
    SELECT A,B,C, NULL, NULL, NULL FROM Rl
    WHERE C NOT IN (SELECT C FROM
          R2)
```

însă toate cele trei SGBD-uri se încăpăţânează să nu o execute.

Revenim la câteva exemple prezentate la joncţiunea externă pe care le rezolvăm prin noua „reţetă" - reuniune/valori vide.

Care sunt valorile facturate şi încasate ale fiecărei facturi?

Soluţia de mai jos (valabilă deopotrivă în DB2, Oracle şi VFP) reuneşte facturile care au măcar o tranşă de încasare cu cele neîncasate deloc.

```
SELECT F. NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA)) / COUNT(DISTINCT I.Codlnc) AS
    Facturat,
    SUM(Transa) / MAX(LF.Linie) AS încasat FROM LINIIFACT LF, PRODUSE P,
FACTURI F, INCASFACT I WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact AND
F.NrFact=I.NrFact GROUP BY F.NrFact
```

```
UNION
   SELECT F. NrFact,
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat,
      0  AS încasat FROM LINIIFACT LF, PRODUSE P, FACTURI F WHERE LF.CodPr = P.CodPr
   AND LF.NrFact = F.NrFact AND F.NrFact NOT IN (SELECT NrFact FROM INCASFACT)
   GROUP BY F.NrFact
```

Să se obţină sporurile de noapte pentru al doilea trimestru al anului 2000, atât lunar, cât şi cumulat.

Trebuie reunite persoanele care au sporul de noapte pe toate cele trei luni cu persoanele care prezintă sporul numai pe câte două luni şi cu persoanele cu sporul pe o singură lună.

Dacă la momentul formulării soluţiei anterioare afirmam că fraza SELECT este supraponderală, prezenta soluţie este pur şi simplu pantagruelică. Cred că fraza următoare (care funcţionează în DB2 şi Oracle) demonstrează fără dubii că joncţiunea externă este o găselniţă grozav de inteligentă şi, pe de altă parte, că scrierea frazelor SQL poate deveni, pe alocuri, o treabă înfricoşătoare.

```
SELECT PERSONAL2.Marca, NumePren,
   SI.SporNoapte AS Spor_Noapte_Aprilie,
   S2.SporNoapte AS Spor_Noapte_Mai,
   S3 . SporNoapte AS Spor_Noapte__Iunie,
   SI. SporNoapte + S2.SporNoapte + S3.SporNoapte AS Spor_Noapte_Trim_II FROM PERSONAL2,
SPORURI SI, SPORURI S2, SPORURI S3 WHERE PERSONAL2.Marca=Sl.Marca AND Sl.An=2000
AND 4=Sl.Luna AND PERSONAL2.Marca=S2.Marca AND S2.An=2000 AND 5=S2.Luna AND
PERS0NAL2.Marca=S3.Marca AND
         S3.An=2000 AND 6 = S3.Luna UNION.
   SELECT PERSONAL2.Marca, NumePren,
      SI.SporNoapte,
      S2.SporNoapte,
      0,
      SI. SporNoapte + S2 . SporNoapte FROM PERSONAL2, SPORURI SI, SPORURI S2 WHERE
   PERSONAL2.Marca=Sl.Marca AND
      51.            An=2000   AND 4 = SI.Luna AND PERSONAL2.Marca=S2.Marca AND
      52.            An=2000   AND 5 = S2.Luna AND PERSONAL2.Marca NOT IN
         (SELECT Marca FROM SPORURI
         WHERE An=2000 AND Luna=6)
UNION
SELECT PERSONAL2.Marca, NumePren,
SI.SporNoapte,
0,
```

```
    S3.SporNoapte,
    SI.SporNoapte + S3.SporNoapte FROM PERS0NAL2, SPORURI SI, SPORURI S3
WHERE PERS0NAL2.Marca=Sl.Marca AND Sl.An=2000 AND
    4  = SI.Luna AND PERS0NAL2.Marca=S3.Marca AND S3.An=2000 AND 6 = S3.Luna
        AND PERS0NAL2.Marca NOT IN (SELECT Marca FROM
        SPORURI
        WHERE An=2000 AND Luna=5)
UNION
SELECT PERS0NAL2.Marca, NumePren,
    0,
    S2.SporNoapte,
    S3.SporNoapte,
    S2.SporNoapte + S3.SporNoapte FROM PERSONAL2, SPORURI S2, SPORURI S3 WHERE
PERSONAL2.Marca=S2.Marca AND S2.An=2000 AND
    5  = S2.Luna AND PERSONAL2.Marca=S3.Marca AND S3.An=2000 AND 6 = S3.Luna
    AND PERSONAL2.Marca NOT IN (SELECT Marca FROM SPORURI
        WHERE An=2000 AND Luna=4)
UNION
    SELECT PERSONAL2.Marca, NumePren,
        SI.SporNoapte,
        0, 0,
        SI.SporNoapte FROM PERSONAL2, SPORURI SI WHERE
    PERSONAL2.Marca=Sl.Marca AND 2000=Sl.An AND 4 = SI.Luna
    AND PERSONAL2.Marca NOT IN (SELECT Marca FROM
    SPORURI
            WHERE An=2000 AND Luna=5)
        AND PERSONAL2.Marca NOT IN (SELECT Marca FROM
            SPORURI
            WHERE An=2000 AND Luna=6)
UNION
SELECT PERSONAL2.Marca, NumePren,
    0,
    S2 . SporNoapte,
    0,
    S2.SporNoapte FROM PERSONAL2, SPORURI S2
```

```
WHERE PERS0NAL2.Marca=S2.Marca AND 2000=S2.An AND 5= S2.Luna AND PERS0NAL2.Marca
NOT IN (SELECT Marca FROM SPORURI
      WHERE An=2000 AND Luna=4)
   AND PERSONAL2.Marca NOT IN (SELECT Marca FROM
      SPORURI                    .
      WHERE An=2000 AND Luna=6)
UNION
   SELECT PERSONAL2.Marca, NumePren,
      0, 0,
      S3.SporNoapte,
      S3.SporNoapte FROM PERSONAL2, SPORURI S3 WHERE
   PERSONAL2.Marca=S3.Marca AND 2000=S3.An AND 6= S3.Luna
   AND PERSONAL2.Marca NOT IN (SELECT Marca FROM
   SPORURI
         WHERE An=2000 AND Luna=4)
      AND PERSONAL2.Marca NOT IN (SELECT Marca FROM
         SPORURI
         WHERE An=2000 AND Luna=5)
UNION
SELECT PERSONAL2.Marca, NumePren,
   0, 0,
   0,
   0
FROM PERSONAL2 WHERE Marca NOT IN (SELECT Marca FROM SPORURI
      WHERE An=2000 AND Luna=4)
   AND Marca NOT IN (SELECT Marca FROM
      SPORURI
      WHERE An=2000 AND Luna=5)
   AND Marca NOT IN (SELECT Marca FROM
      SPORURI
      WHERE An=2000 AND Luna=6)
ORDER BY NumePren
```

Cred că singura mângâiere este că am scris, de voie, de nevoie, cea mai lungă frază SELECT de până acum...

Tot cu ajutorul operatorului IN (şi NOT IN) se poate aborda şi „problema" *diviziunii* relaţionale în SQL (parcă vă aud spunând: „Asta ne mai lipsea acuma!"). Succesiunea de paşi prezentată în figura 2.28 se realizează relativ simplu prin soluţia SQL-92/DB2:

```
SELECT   X   FROM   Rl
EXCEPT
   SELECT DISTINCT Rl.X FROM Rl, R2
   WHERE (Rl.X, R2.Y) NOT IN (SELECT X, Y FROM Rl)
```

în Oracle este necesară înlocuirea operatorului EXCEPT cu MINUS. Cu VFP e o poveste mai lungă. Cum aici nu există nici EXCEPT, nici MINUS, sau ceva similar, am încercat varianta:

```
SELECT 28 ;
FROM Rl, R2 ;
WHERE R1.X+R2.Y NOT IN ;
   (SELECT X+Y ;
   FROM Rl)
```

Mesajul primit a fost unul curat, limpede, tonic şi explicit ca al unui funcţionar de la ghişeele Oficiilor Forţelor de Muncă: SQL: Queries of this type are not supported (Error 1814). Ce-i de făcut? Salvarea vine tot de Ia joncţiunea externă:

```
SELECT DISTINCT X ;
FROM Rl ;
WHERE X+' ' NOT IN ;
   (SELECT Rl_l.X + NVL(Rl_2.X,' ') ;
   FROM Rl Rl_l ;
      INNER JOIN R2 R2_l ON 1=1 ;
      LEFT OUTER JOIN Rl Rl_2 ;
         ON Rl_l.X=R1_2.X AND R2_l.Y=R1_2.Y)
```

Pseudojoncţiunea internă este de fapt un produs cartezian, deoarece condiţia de joncţiune este 1= 1. Subconsultarea extrage „icşii" care au „goluri", iar fraza SELECT principală efectuează scăderea lor din „icşii" tabelei Rl.

Aflaţi clienţii pentru care există cel puţin câte o factură emisă în fiecare zi.

Acesta este exemplul 22 din algebra relaţională (figura 2.27) pentru ilustrarea operatorului diviziune. Urmăm logica pe care tocmai am prezentat-o.

```
          WHERE C.CodCl=Fl.CodCl AND
             (C.CodCl, F2.DataFact) NOT IN (SELECT C.CodCl,
                DataFact FROM CLIENŢI C, FACTURI F WHERE
                C.CodCl=F.CodCl))
```

- Soluţia VFP:

```
SELECT DISTINCT DenCl ;
FROM CLIENŢI ;
   INNER JOIN FACTURI ON CLIENŢI.CodCl=FACTURI.CodCl ; WHERE
STR(CLIENTI.CodCl,6}+STR(0,6) NOT IN ;
   (SELECT STR (FI .CodCl, 6-) +STR(NVL (F3 .CodCl, 0) , 6)   ;
   FROM FACTURI FI INNER JOIN FACTURI F2 ON 1=1 ;
                          LEFT OUTER JOIN FACTURI F3 ON ;
      FI.CodCl=F3.CodCl AND F2.DataFact=F3.DataFact)
```

In ce zile s-au vândut şi produsul cu denumirea „ Produs I ", şi cel cu denumirea „ Produs 2 "? Este exemplul 23 din acelaşi capitol 2.

- Soluţia DB2 (şi Oracle, înlocuind `EXCEPT` cu `MINUS`):

```
SELECT DataFact
      FROM FACTURI F, LINIIFACT LF, PRODUSE P WHERE F.NrFact=LF.NrFact AND
            LF.CodPr=P.CodPr AND DenPr IN ('Produs 1', 'Produs 2')
EXCEPT                                                    ;
   SELECT DISTINCT F.DataFact
   FROM FACTURI F, LINIIFACT LF, PRODUSE PI, PRODUSE P2 WHERE
   F.NrFact=LF.NrFact AND LF.CodPr=Pl.CodPr AND PI.DenPr IN ('Produs 1', 'Produs
   2')
      AND P2.DenPr IN ('Produs 1', 'Produs 2')
```

---

28 Soluţia DB2 (şi Oracle, cu MINUS-ul de rigoare):

```
SELECT DenCl FROM CLIENŢI WHERE CodCI IN (SELECT CodCI FROM CLIENŢI C EXCEPT
    SELECT DISTINCT C.CodCI
    FROM CLIENŢI C, FACTURI FI, FACTURI F2
```

```
                    AND (F.DataFact, P2.CodPr) NOT IN
        (SELECT DISTINCT DataFact, LF.CodPr FROM FACTURI F, LINIIFACT LF,
        PRODUSE P WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr AND
        DenPr IN ('Produs 1', 'Produs 2' ) )
```

- Soluţia VFP:

```
SELECT DISTINCT DataFact ;
FROM FACTURI ;
   INNER JOIN LINIIFACT ON FACTURI.NrFact=LINIIFACT.NrFact INNER JOIN
   PRODUSE ON LINIIFACT.CodPr=PRODUSE.CodPr AND DenPr IN (1 Produs 1', 'Produs
   2')                                                    ;
WHERE DTOC(DataFact)+DTOC({//}) NOT IN ;
   (SELECT DISTINCT DTOC(FI.DataFact)+;
                              DTOC(NVL(F2.DataFact, {//}));
   FROM FACTURI FI INNER JOIN PRODUSE PI ;
         ON PI.DenPr IN ('Produs 1','Produs 2')              ;
      LEFT OUTER JOIN (LINIIFACT LF2 INNER JOIN FACTURI F2
```

```
      ON LF2.NrFact=F2.NrFact) ;
         ON FI.DataFact=F2.DataFact AND PI.CodPr=LF2.CodPr)
```

Prezenta soluţie mi-a prilejuit descoperirea unei facilităţi pe care alţii, probabil, o ştiau de mult. Şi în VFP este posibilă joncţionarea unei tabele cu rezultatul unei alte joncţiuni, precedenţa joncţionărilor fiind indicată cu ajutorul parantezelor. Fără această opţiune, lucrurile ar fi avut un aer mult mai grav.

încheiem paragraful dedicat operatorului IN cu un exemplu mai puţin „colţuros", dar suficient de interesant.

Să se afişeze câte facturi sunt: neîncasate deloc, încasate parţial şi încasate total?
Soluţia reuneşte facturile încasate total cu facturile încasate parţial şi cu facturile neîncasate deloc.

```
SELECT 'încasate TOTAL' AS Situatiune, COUNT(*) AS Nr FROM FACTURI WHERE
NrFact IN (SELECT F.Nrfact
   FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I WHERE LF.CodPr =
   P.CodPr AND LF.NrFact = F.NrFact AND F.NrFact=I.NrFact GROUP BY F.NrFact
   HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) /
      COUNT(DISTINCT I.Codlnc) =
            SUM(Transa) / MAX(LF.Linie))
UNION
   SELECT 'încasate parţial', COUNT(*)
   FROM FACTURI WHERE NrFact IN (SELECT F.NrFact
      FROM LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I WHERE LF.CodPr =
      P.CodPr AND LF.NrFact = F.NrFact AND F.NrFact=I.NrFact GROUP BY F.NrFact
      HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) /
         COUNT(DISTINCT I.Codlnc) >
            SUM(Transa) / MAX(LF.Linie))
UNION
SELECT 'Neincasate deloc', COUNT(*)
FROM FACTURI WHERE
   NrFact IN
      (SELECT NrFact FROM LINIIFACT)
   AND
   NrFact NOT IN (SELECT NrFact FROM INCASFACT)
```

SITUATIUNE	NR
încasate TOTAL	4
încasate parţial	2
Neincasate deloc	5

Figura 5.26. Numărul facturilor fără nici o tranşă de încasare, încasate parţial şi încasate total

Până acum subconsultările au fost conectate la fraza SELECT superioară exclusiv prin operatorul IN. în paragraful următor vom vedea că pentru (sub)interogările comparative pot fi întrebuinţaţi ALL, SOME, ANY. Atunci când rezultatul unei subconsultări se concretizează într-o tabelă cu o singură coloană şi o singură linie, corelarea poate fi făcută cu operatorii de comparaţie obişnuiţi: = , >, >=, <, <=. Vom ilustra această facilitate prin câteva exemple.

Care este cel mai mare preţ unitar la care s-a vândut un produs?

```
SELECT MAX(PretUnit)
FROM LINIIFACT
```

Care este cel mai mare preţ unitar şi care este produsul, precum şi factura unde se înregistrează respectivul preţ maxim?

```
SELECT NrFact, DenPr, PretUnit
FROM LINIIFACT LF, PRODUSE P
WHERE LF.CodPr=P.CodPr AND PretUnit =
    (SELECT MAX(PretUnit)
    FROM LINIIFACT)
```

Care sunt cele mai mari două preţuri unitare de vânzare, care sunt produsele şi facturile pentru care se înregistrează respectivele preţuri maxime?

```
SELECT NrFact, DenPr, PretUnit
FROM LINIIFACT LF, PRODUSE P
WHERE LF.CodPr=P.CodPr AND PretUnit >=
    (SELECT MAX(PretUnit)
    FROM LINIIFACT WHERE PretUnit <
        (SELECT MAX(PretUnit)
        FROM LINIIFACT)
    )
```

Pentru a înţelege mecanismul interogării de mai sus, pornim de la SELECT-ul „cel mai de jos”. SELECT MAX (PretUnit) FROM LINIIFACT extrage preţul unitar maxim din tabela LINIIFACT. Subconsultarea superioară, (SELECT MAX (PretUnit) FROM LINIIFACT WHERE PretUnit < (...ultima subconsultare...), determină al doilea preţ unitar din LINIIFACT. SELECT-ul principal afişează toate preţurile unitare mai mari sau egale cu penultimul.

Care sunt cele mai mari cinci preţuri unitare de vânzare, produsele şi facturile în care apar cele cinci preţuri maxime?

Aici voiam, de fapt, să ajungem:

```
SELECT NrFact, DenPr, PretUnit FROM LINIIFACT INNER JOIN PRODUSE ON
LINIIFACT.CodPr=PRODUSE.CodPr WHERE PretUnit >
    (SELECT MAX(PretUnit)
    FROM LINIIFACT WHERE PretUnit <
            (SELECT MAX(PretUnit)
        FROM LINIIFACT WHERE PretUnit <
            (SELECT MAX(PretUnit)
            FROM LINIIFACT WHERE PretUnit <
                (SELECT MAX(PretUnit)
                FROM LINIIFACT WHERE PretUnit <
                    (SELECT MAX(PretUnit)
                    FROM LINIIFACT WHERE PretUnit <
                        (SELECT MAX(PretUnit)
                        FROM LINIIFACT)
                    )
                )
            )
    ) )
ORDER BY PretUnit DESC
```

Celor care nu au reuşit să fie impresionaţi de această ultimă variantă, le sugerez să încerce cu primele 10, 20 ş.a.m.d. preţuri unitare. Revenim însă la soluţia prezentată; iată rezultatul acesteia (figura 5.27).

NRFACT DENPR PRETUNIT

1114	Produs 4	15820	
1119		: Produs 4	14000
1120		; Produs 2	11200
1118		Produs 2	;11000
1119		Produs 2	10900

Figura 5.27. Cele mai mari cinci preţuri unitare

Firește, la atâta amar de niveluri de interogare, VFP capotează. Avem însă la îndemână o soluţie neverosimil de simplă, bazată de clauza TOP:

```
SELECT TOP 5 NrFact, DenPr, PretUnit ;
FROM LINIIFACT INNER JOIN PRODUSE ON
    LINIIFACT.CodPr=PRODUSE.CodPr ;
ORDER BY PretUnit DESC
```

Figura 5.29. Produse cu cel puţin un preţ unitar superior oricărui preţ unitar al „Produsului 1"

Liniile sunt ordonate descrescător după preţul unitar, iar în rezultate sunt extrase primele cinci (prin `TOP 5)`.

5.5. Operatorii ALL, SOME, ANY

Cei trei operatori prezentaţi în acest paragraf sunt grozav de utili în interogările cu iz „cantitativist" mai pronunţat, deoarece permit utilizarea unui predicat de comparaţie care este aplicat rezultatului unei subconsultări, predicat bazat pe unul dintre operatorii: =, <, >, <=, >=, < >sau#.

Dacă, în cea mai mare parte a cazurilor de până acum, se compara un atribut (sau rezultatul unei expresii/funcţii) cu o constantă, `ALL`, `SOME` şi `ANY` permit compararea valorii atributului/funcţiei/expresiei cu un set de tupluri (absamblu de linii) extras printr-o subconsultare.

Care sunt produsele vândute la preţuri unitare superioare oricărui preţ unitar la care a fost vândut „Produs 1 "?

Soluţia acestei probleme este valabilă în SQL-92 şi în toate cele trei SGBD-uri:

```
SELECT DISTINCT DenPr FROM PRODUSE P,
LINIIFACT LF WHERE P.CodPr=LF.CodPr AND
PretUnit > ALL (SELECT DISTINCT PretUnit FROM
PRODUSE P, LINIIFACT LF
   WHERE P.CodPr=LF.CodPr AND DenPr ='Produs 1')
```

Ca orice interogare pe două niveluri, ostilităţile se derulează în doi paşi. Mai întâi se execută subconsultarea şi se obţine o tabelă intermediară în care se găsesc toate preţurile unitare la care a fost vândut, în decursul istoriei, *Produs 1* - vezi figura 5.28.

PRETUNIT
9300
9500
10000

Figura 5.28. Rezultatul subconsultării - preţurile unitare pentru „Produs 1"

Cum operatorul de conexiune a frazei `SELECT` principale cu subconsultarea este > ALL, din joncţiunea tabelelor PRODUSE şi LINIIFACT vor fi extrase numai liniile care au valoarea atributului `PretUnit` mai mare decât toate valorile din figura 5.28. Aşa încât rezultatul final se prezintă ca în figura 5.29.

DENPR
Produs 2
Produs 4

Care sunt produsele vândute la preţuri unitare superioare măcar unui preţ unitar al „ Produsului 1 ”?

Este genul de situaţii în care se foloseşte SOME sau ANY (au aceeaşi funcţiune).

```
SELECT DISTINCT DenPr FROM PRODUSE P, LINIIFACT LF WHERE
P.CodPr=LF.CodPr AND PretUnit > ANY (SELECT DISTINCT PretUnit FROM
PRODUSE P, LINIIFACT LF
    WHERE P.CodPr=LF.CodPr AND DenPr ='Produs 1')
```

Rezultatul este cel din figura 5.30, deoarece, spre deosebire de ALL, şi ANY (şi SOME) selectează liniile pentru care preţul unitar este mai mare decât măcar una dintre valorile obţinute prin subconsultare.

DENPR
Produs 1
Produs 2
Produs 4

Figura 5.30. Produse cu cel puţin un preţ unitar superior măcar unui preţ unitar al „Produsului 1“

Operatorul =ANY este echivalent cu IN.

Câţi alţi angajaţi au salariul tarifar egal cu al ANGAJAT 2?

Soluţia:

```
SELECT COUNT(*) - 1 AS Nr FROM PERSONAL2 WHERE SalTarifar IN
(SELECT Saltarifar FROM PERSONAL2 WHERE NumePren='ANGAJAT 2')
```

este echivalentă cu:

```
SELECT COUNT(*) - 1 AS Nr FROM PERSONAL2 WHERE SalTarifar =ANY
(SELECT Saltarifar FROM PERSONAL2 WHERE NumePren='ANGAJAT 2')
```

Folosirea unuia din cei trei operatori nu este obligatorie atunci când subconsultarea conţine o funcţie-agregat (ce întoarce o valoare dintr-un ansamblu de tupluri) - MIN, MAX, COUNT, SUM, AVG.

Care este ultima factură întocmită (factura cea mai recentă) şi data la care a fost emisă?

Oricare dintre variantele următoare funcţionează şi obţine valorile din figura 5.31.

DATAFACT	jULTIMAFACTURA
07-08-2000	'1122

```
SELECT DataFact, NrFact AS UltimaFactura FROM FACTURI WHERE NrFact IN
    (SELECT MAX(NrFact)
                    FROM FACTURI)

SELECT DataFact, NrFact AS UltimaFactura FROM FACTURI WHERE NrFact =
    (SELECT MAX(NrFact)
                    FROM FACTURI)

SELECT DataFact, NrFact AS UltimaFactura FROM FACTURI WHERE NrFact =ANY
    (SELECT MAX(NrFact)
                    FROM FACTURI)

SELECT DataFact, NrFact AS UltimaFactura FROM FACTURI WHERE NrFact =ALL
```

Figura 5.31. Ultima factură şi ziua în care a fost întocmită

```
(SELECT MAX(NrFact)
                    FROM FACTURI)
```

Fără funcţia MAX, soluţia este:

```
SELECT DataFact, NrFact AS UltimaFactura FROM FACTURI WHERE NrFact >= ALL
(SELECT NrFact FROM FACTURI)
```

5.6. Subconsultări în clauza HAVING

Predicatele incluse în clauza HAVING ale interogărilor de până acum compară o expresie cu o constantă. In cele ce urmează vom discuta despre includerea în clauza HAVING a subconsultărilor. Din păcate, deoarece această facilitate nu a fost încă implementată (până la versiunea VFP6) în nucleul SQL al acestui produs, trimiterile la Visual FoxPro vor fi mai lapidare în acest paragraf.

Revenim asupra unei probleme formulate în ultima parte a paragrafului 4.6:

Care sunt zilele în care s-au emis mai multe facturi decât pe 2 august 2000?
Se poate formula o soluţie bazată pe subconsultări în clauza HAVING (prezenta este redactată în DB2 - în Oracle este necesară funcţia de conversie TO_DATE):

```
SELECT DataFact AS Zi, COUNT(NrFact) AS Nr_Facturilor FROM FACTURI GROUP BY
DataFact HAVING COUNT(NrFact) >
    (SELECT COUNT (NrF'act)
    FROM FACTURI
    WHERE DataFact - '08/02/2000')
```

Care este ziua în care s-au emis cele mai multe facturi?

```
SELECT DataFact, COUNT(*) AS Nr Facturilor FROM FACTURI GROUP BY DataFact
HAVING COUNT(*) >= ALL (SELECT COUNT(*)
    FROM FACTURI GROUP BY DataFact)
```

Subconsultarea calculează numărul de facturi corespunzător fiecărei zile. Predicatul clauzei HAVING compară numărul de facturi al fiecărei zile cu toate valorile extrase de subconsultare. Se obţine tabela din figura 5.32.

DATAFACT	NR.FACTURILOR
01-AUG-00	4
07-AUG-00	4

Figura 5.32. Zilele în care s-au emis cele mai multe facturi

Se cuvine de adăugat că Oracle mai permite şi o soluţie bazată pe incluziunea unei funcţii în alta:

```
SELECT DataFact, COUNT(*) AS Nr Facturilor FROM FACTURI GROUP BY DataFact
HAVING COUNT(*) =
    (SELECT MAX(COUNT(*))
    FROM FACTURI GROUP BY DataFact)
```

Am fi nedrepţi cu VFP dacă nu am prezenta o soluţie care mie, unuia, îmi place pentru simplitate. Iat-o:

```
SELECT TOP 1 DataFact, COUNT(*) AS Nr ;                                    '
FROM FACTURI ;
GROUP BY DataFact ;
ORDER BY Nr DESC
```

Surprinzător pentru unii, soluţia funcţionează. ORDER BY se aplică grupurilor, iar ordonarea

împreună cu opţiunea TOP extrag rezultatul corect.

Care este clientul care a cumpărat cele mai multe produse?

- soluţie SQL-92/DB2/Oracle:

```
SELECT DenCl, COUNT(DISTINCT CodPr) AS CiteProduse FROM CLIENŢI C, FACTURI F,
LINIIFACT LF WHERE C.CodCl=F.CodCI AND F.NrFact= LF.NrFact
    GROUP BY DenCl
          HAVING COUNT(DISTINCT CodPr) >= ALL (SELECT COUNT(DISTINCT CodPr)
        FROM FACTURI F, LINIIFACT LF WHERE F.NrFact= LF.NrFact GROUP
        BY CodCl)
```

DENCL	CITEPRODUSE
Client 3 SRL	4

Figura 5.33. Clientul care a cumpărat cele mai multe produse

- alta „Oracle only” (COUNT în MAX):

```
SELECT DenCl, COUNT(DISTINCT CodPr) AS CiteProduse FROM CLIENŢI
C, FACTURI F, LINIIFACT LF WHERE C.CodCl=F.CodCl AND F.NrFact=
LF.NrFact GROUP BY DenCl
HAVING COUNT(DISTINCT CodPr) =
    (SELECT MAX(COUNT(DISTINCT CodPr))
    FROM FACTURI F, LINIIFACT LF WHERE F.NrFact= LF.NrFact GROUP
    BY CodCl)
```

- şi una VFP (clauza TOP):

```
SELECT TOP 1 DenCl, COUNT(DISTINCT CodPr) AS CiteProduse ; FROM CLIENŢI C
INNER JOIN FACTURI F ON C.CodCl=F.CodCl ;
    INNER JOIN LINIIFACT LF ON F.NrFact= LF.NrFact ;
GROUP BY DenCl ;
ORDER BY CiteProduse DESC
```

Care este compartimentul cu cea mai bună medie a salariilor tarifare?

Se exclude din discuţie compartimentul DIRECŢIUNE în care apare, singur şi ferice, directorul general. •

```
SELECT Compart, AVG(SalTarifar) AS Medie_Sal FROM
PERSONAL2
WHERE Compart <> 'DIRECŢIUNE'
GROUP BY Compart HAVING AVG(SalTarifar)
>= ALL (SELECT AVG(SalTarifar)
    FROM PERSONAL2
                        WHERE Compart O 'DIRECŢIUNE'
    GROUP BY Compart)
```

COMPART	MEDIE_SAL
RESURSE UMANE	5500000

Pentru aducerea aminte a structurilor alternative, putem folosi şi variantele:

- SQL-92/DB2:

```
SELECT Compart, AVG (CASE
        WHEN Compart <> 'DIRECŢIUNE'
```

```
        THEN SalTarifar
ELSE 0 END) AS Medie_Sal
FROM PERSONAL2 GROUP
BY Compart HAVING AVG
(CASE
      WHEN Compart O 'DIRECŢIUNE'
        THEN SalTarifar
ELSE 0 END) >= ALL
(SELECT AVG (CASE
      WHEN Compart <> 'DIRECŢIUNE'
        THEN SalTarifar
ELSE 0 END)
   FROM PERS0NAL2 GROUP BY Compart)
```

- şi Oracle:

```
SELECT Compart, ROUND(AVG (
   DECODE (Compart, 'DIRECŢIUNE', 0, SalTarifar)
   ),0) AS Medie_Sal FROM PERSONAL2 GROUP BY Compart
HAVING
   AVG (DECODE (Compart, 'DIRECŢIUNE', 0, SalTarifar))
   >= ALL
      (SELECT AVG (DECODE (Compart, 'DIRECŢIUNE', 0,
            SalTarifar))
       FROM PERSONAL2 GROUP BY Compart)
```

Funcţia ROUND a fost necesară pentru afişarea numai a părţii întregi din medie. De data aceasta (şi următoarele) trecem peste varianta Oracle cu funcţie inclusă în altă funcţie.

- în fine, varianta VFP:

```
SELECT TOP 1 Compart, AVG(SalTarifar) AS Medie_Sal ;
FROM PERSONAL2 ;
WHERE Compart O 'DIRECŢIUNE' ;
GROUP BY Compart ;
ORDER BY Medie Sal DESC
```

Care este judeţul în care berea s-a vândut cel mai bine?

în tabela PRODUSE există un atribut care reprezintă grupa în care se încadrează produsul respectiv. Berea este una dintre grupele îndrăgite.

```
SELECT Judeţ, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari_Bere
FROM JUDEŢE J, LOCALITATI L, CLIENŢI C, FACTURI F,
LINIIFACT LF, PRODUSE P
WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND C.CodCl=F.CodCl AND
   F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr AND Grupa='Bere'
GROUP BY Judeţ
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA))
   >= ALL
(SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
FROM JUDEŢE J, LOCALITATI L, CLIENŢI C, FACTURI F,
     LINIIFACT LF, PRODUSE P WHERE J.Jud^L.Jud AND
L.CodPost=C.CodPost AND
     C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND
     LF.CodPr=P.CodPr AND       Grupa='Bere'
GROUP BY Judeţ)
```

VINZARI.BERE	JUDEŢ
	Iasi 7646940

Figura 5.35. Judeţul cu cea mai bună vânzare a produselor din grupa Bere

Să nu uităm soluţia funcţională în VFP, cuceritoare prin simplitate:

```
SELECT TOP 1 Judeţ, SUM(Cantitate * PretUnit * (1+ProcTVA))\ AS
   Vinzari_Bere ;
FROM JUDEŢE J INNER JOIN LOCALITATI L ON J.Jud=L.Jud ;
   INNER JOIN   CLIENŢI  C  ON L.CodPost=C.CodPost     ;
   INNER JOIN   FACTURI  F  ON C.CodCl=F.CodCl ;
   INNER JOIN   LINIIFACT  LF ON  F.NrFact=LF.NrFact    ;
   INNER JOIN   PRODUSE  P  ON LF.CodPr=P.CodPr ;
WHERE Grupa='Bere' ;
GROUP BY Judeţ ;
ORDER BY Vinzari_Bere DESC
```

Care sunt clienţii cu valoarea vânzărilor peste medie?

```
SELECT DenCl AS Client,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
LINIIFACT LF, PRODUSE P, FACTURI F, CLIENŢI C WHERE P.CodPr =
LF.CodPr AND LF.NrFact = F.NrFact AND F.CodCl=C.CodCl GROUP BY
DenCl
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) >=
   (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)) /
```

```
COUNT(DISTINCT F.CodCI)
      FROM LINIIFACT LF, PRODUSE P, FACTURI F
           WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact)
```

CLIENT	VINZARI
Client 1 SRL	12490240
Client 3 SRL	7242935
Client 4	5438300
Client 6 SA	6786570

Figura 5.36. Clienţi cu vânzări peste medie

Extrageţi factura cu valoarea imediat peste cea medie.

```
SELECT F.NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS
Valoare FROM LINIIFACT LF, PRODUSE P, FACTURI F WHERE P.CodPr
= LF.CodPr AND LF.NrFact = F.NrFact GROUP BY F.NrFact HAVING
   SUM(Cantitate * PretUnit * (1+ProcTVA)) <= ALL (SELECT
      SUM(Cantitate * PretUnit * (1+ProcTVA))
      FROM LINIIFACT LF, PRODUSE P, FACTURI F
      WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact
      GROUP BY F.NrFact
      HAVING
            SUM(Cantitate * PretUnit * (1+ProcTVA)) >
         (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)) /
            COUNT(DISTINCT F.NrFact)
         FROM LINIIFACT LF, PRODUSE P, FACTURI F WHERE P.CodPr =
      LF.CodPr AND LF.NrFact = F.NrFact)) AND
      SUM(Cantitate * PretUnit * (1+ProcTVA)) >
         (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)) /
            COUNT(DISTINCT F.NrFact)
         FROM LINIIFACT LF, PRODUSE P, FACTURI F
         WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact)
```

NRFACT	VALOARE
1111	5399625

Figura 5.37. Factura cu cea mai mică valoare peste medie Elementul de noutate al acestei interogări este includerea, într-o consultare ce prezintă clauza HAVING, a unei alte subconsultări în care apare, de asemenea, HAVING. Pentru VFP nu am nici o variantă la îndemână.

Care este judeţul cu vânzări imediat superioare judeţului Neamţ?

```
SELECT Judeţ, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS
   Vinzari
FROM JUDEŢE J, LOCALITATI L, CLIENŢI C, FACTURI F, LINIIFACT
LF, PRODUSE P WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
GROUP BY Judeţ
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) <= ALL (SELECT
   SUM(Cantitate * PretUnit * (1+ProcTVA))
   FROM JUDEŢE J, LOCALITATI L, CLIENŢI C, FACTURI F,
   LINIIFACT LF, PRODUSE P WHERE J.Jud=L.Jud AND
   L.CodPost=C.CodPost AND C.CodCl=F.CodCl AND
   F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr GROUP BY Judeţ
   HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) >
         (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)) FROM
         JUDEŢE J, LOCALITATI L, CLIENŢI C,
            FACTURI F,LINIIFACT LF, PRODUSE P WHERE J.Jud=L.Jud
         AND L.CodPost=C.CodPost AND C.CodCl=F.CodCl AND
         F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr AND
         Judet='Neamţ'))
   AND
      SUM(Cantitate * PretUnit * (1+ProcTVA)) >
      (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)) FROM JUDEŢE
      J, LOCALITATI L, CLIENŢI C, FACTURI F, LINIIFACT LF,
      PRODUSE P WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
      C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
      AND Judet='Neamţ')
```

JUDEŢ	VINZARI
Vaslui	7242935

Figura 5.38. Judeţul cu vânzări imediat superioare judeţului Neamţ

Care este clientul cel mai mare datornic?

Este, cred, cea mai grea interogare de până acum. După cum am văzut atunci când am calculat pentru fiecare factură valorile facturate şi încasate, la joncţiunea externă a „părţii de facturare" cu „partea de încasare", fiecare tranşă de încasare se repetă în funcţie de numărul de linii ce compun factura respectivă, iar fiecare linie a facturii se repetă în funcţie de numărul de tranşe de încasare. Trucul utilizat pentru a o scoate la capăt consta în împărţirea totalului tranşelor de încasare la numărul liniilor din factură, obţinând astfel valoarea încasată a facturii şi, pe de altă parte, împărţirea totalului valorilor liniilor din facturi la numărul tranşelor de încasare.

Pentru problema de faţă trebuie găsită o altă soluţie, deoarece gruparea nu o facem la nivel de factură, ci la nivel de client. Numărul de linii din facturi şi numărul de tranşe nu mai au nici o relevanţă la gruparea după client.

Pentru încasări, ar fi o idee. Deoarece o tranşă se repetă pentru fiecare linie a facturii, putem elimina din SUM toate tranşele pentru care `Linie > 1`. încercăm ceva în DB2:

```
SELECT DenCl AS Client,
    SUM (
        CASE WHEN LF.Linie > 1
                    THEN NULL ELSE VALUE(Transa,0)
        END) AS Valoare_Incasata
FROM PRODUSE P
```

```
INNER JOIN LINIIFACT LF ON P.CodPr = LF.CodPr INNER JOIN FACTURI F ON
LF.NrFact = F.NrFact INNER JOIN CLIENŢI C ON F.CodCI = C.CodCI LEFT
OUTER JOIN INCASFACT I ON F.Nrfact=I.NrfACT GROUP BY DenCl
```

Rezultatul din figura 5.39 ne încredinţează că ideea n-a fost chiar rea.

CLIENT	VALOARE_INCASATA
Clienţi SRL	10504432
Client 2 SA	1160250
Client 3 SRL	0
Client 4	0
Client 5 SRL	487705
Cliente SA	0
Client 7 SRL	0

Figura 5.39. încasările pe clienţi (DB2)

Cu valoarea facturată însă, lucrurile sunt mai complicate. Fiecare linie din factură se repetă de atâtea ori, în funcţie de câte tranşe de încasare există pentru factura respectivă. Cum gruparea se face pe clienţi, numărul tranşelor este irelevant. Trebuie însumate valorile *distincte* ale facturii şi liniei. Este adevărat, clauza DISTINCT poate fi folosită şi cu SUM. Dar este aproape sigur vă vom avea, la un moment dat, două linii, în aceeaşi factură, cu aceeaşi valoare. E musai de însumat valorile expresiei pentru combinaţii distincte (NrFact, Linie). Confirmând vocaţia de popor de improvizatori (n-am zis *cârpaci,* să nu supăr adevăraţii patrioţi), recurgem la un truc ieftin, dar, vorba dlui Graur (comentatorul), năucitor!

```
SELECT DenCl,
    SUM( DISTINCT
    1000000000000000 * LF.NrFact +                    -
    1000000000000000 * LF.Linie +
    Cantitate * PretUnit * (1+ProcTVA))
        AS Valoare_Facturata FROM
PRODUSE P
    INNER JOIN LINIIFACT LF ON P.CodPr = LF.CodPr INNER
JOIN FACTURI F ON LF.NrFact = F.NrFact INNER JOIN CLIENŢI
C ON F.CodCI = C.CodCI LEFT OUTER JOIN INCASFACT I ON
F.Nrfact=I.NrfACT GROUP BY DenCl
```

Nu putem discuta detalii înainte de a vedea ce a putut fi obţinut din interogare - figura 5.40.

DENCL	VALOARE_FACTURATA
Clienţi SRL	1005200000012490240.00
Client 2 SA	1114000000001160250.00
Client 3 SRL	4486000000007242935.00
Client 4	2245000000005438300.00
Client 5 SRL	2227000000001337560.00
Client 6 SA	3348000000006786570.00
Client 7 SRL	1117000000001383375.00

Figura 5.40. Improvizaţie pentru calculul valorii vânzărilor pe clienţi (DB2)

Expresia de calcul a valorii facturate introduce, pe lângă Cantitate, PretUnit şi ProcTVA, şi NrFact şi Linie. Ultimele două atribute sunt înmulţite cu constante foarte mari (1 biliard, respectiv 10 bilioane), suficient de mari pentru ca valoarea propriu-zisă a facturilor să nu fie „afectată". Este sigur că valorile obţinute pentru fiecare client au fost calculate luându-se în calcul o singură dată fiecare linie dintr-o factură. Iar dacă privim în rezultat partea din dreapta fiecărui număr (după cele şase-şapte zerouri), observăm că acelea sunt chiar valorile facturate care ne interesează. Prin urmare, nu ne mai rămâne decât să extragem acea parte:

```
SELECT DenCl,
    CAST (
        SUBSTR (
            CAST (
                SUM (
                    DISTINCT 1000000000000000 *
                        LF.NrFactt
                            100000000000000 *
                        LF.Linie+
                                Cantitate * PretUnit * (1+ProcTVA)
                    )
                AS CHAR(35)
                    ) ,17,19
            )
        AS DECIMAL(19,2)
        )
        AS Valoare_Facturata FROM
PRODUSE P
    INNER    JOIN LINIIFACT   LF                ON P.CodPr = LF.CodPr
    INNER    JOIN FACTURI F   ON                  LF.NrFact = F.NrFact
    INNER    JOIN CLIENŢI C   ON                   F.CodCl = C.CodCl
    LEFT OUTER    JOIN INCASFACT I ON F.Nrfact=I.NrfACT
GROUP BY DenCl
```

Obţinem ceea ce doream, adică valoarea vânzărilor pe clienţi, după cum o arată figura 5.41.

DENCL	VALOARE_FACTURATA
Clienţi SRL	12490240.00
Client 2 SA	1160250.00
Client 3 SRL	7242935.00
Client 4	5438300.00
Client 5 SRL	1337560.00
Client 6 SA	6786570.00
Client 7 SRL	1383375.00

Figura 5.41. Valoarea vânzărilor, pe clienţi (DB2)

în final, soluţia SQL-92/DB2 la problema pusă (şi răspunsul în figura 5.42):

```
SELECT DenCl,
    CAST (SUBSTR (CAST (SUM (
    DISTINCT 1000000000000000 * LF.NrFact+ 100000000000000 * LF.Linie+
          Cantitate *       PretUnit *      (1+ProcTVA) )
       AS CHAR(35) )     ,17,19     ) AS DECIMAL(19,2)          )
       AS Valoare_Facturata,
      SUM (CASE WHEN LF.Linie > 1 THEN NULL ELSE VALUE(Transa,0)
                          END) AS Valoare_Incasata,
    CAST (SUBSTR (CAST (SUM (
          DISTINCT 1000000000000000 * LF.NrFact+
          100000000000000 * LF.Liniet Cantitate * PretUnit *
          (1+ProcTVA) )
       AS CHAR(35) )     ,17,19     ) AS DECIMAL(19,2)        ) -
    SUM (CASE WHEN LF.Linie     > 1  THEN NULL ELSE
       VALUE(Transa,0) END)
    AS Rest_De_Incasat FROM
PRODUSE P
    INNER JOIN LINIIFACT LF ON P.CodPr = LF.CodPr INNER JOIN
FACTURI F ON LF.NrFact = F.NrFact INNER JOIN CLIENŢI C ON F.CodCI
= C.CodCI LEFT OUTER JOIN INCASFACT I ON F.NrFact=I.NrFact GROUP
BY DenCl HAVING
    CAST (SUBSTR (CAST (SUM      (
    DISTINCT 1000000000000000 * LF.NrFact+ 100000000000000 * LF.Linie+
          Cantitate * PretUnit * (1+ProcTVA) )
       AS CHAR(35) ) ,17,19 ) AS DECIMAL(19,2) ) - SUM (CASE
    WHEN LF.Linie > 1 THEN NULL ELSE VALUE(Transa,0) END)
>= ALL
(SELECT CAST (SUBSTR (CAST (SUM (
          DISTINCT 1000000000000000 * LF.NrFact+
             100000000000000 * LF.Linie+
          Cantitate * PretUnit * (1+ProcTVA) )
       AS CHAR(35) ) ,17,19 ) AS DECIMAL(19,2) ) - SUM (CASE
    WHEN LF.Linie > 1 THEN NULL ELSE VALUE(Transa,0) END)
FROM PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr = LF.CodPr INNER JOIN
FACTURI F ON LF.NrFact = F.NrFact INNER JOIN CLIENŢI C ON
F.CodCI = C.CodCI LEFT OUTER JOIN INCASFACT I ON
F.NrFact=I.NrFact GROUP BY DenCl)
```

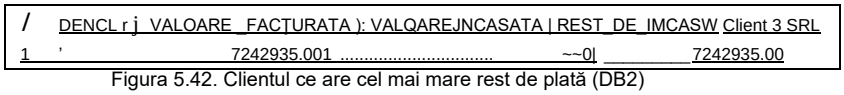


Figura 5.42. Clientul ce are cel mai mare rest de plată (DB2)

Varianta Oracle pentru obţinerea aceluiaşi rezultat este:

```
SELECT DenCl,
     TO_NUMBER( SUBSTR ( TO_CHAR (
     SUM(DISTINCT 1000000000000000 *
            LF.NrFact+100000000000000 *
         LF.Linie+Cantitate * PretUnit * (1+ProcTVA)),
     '99999999999999999999999999'), 14,15))
     AS Valoare_Facturata,
     SUM (DECODE (LF.Linie, 1, NVL(Transa,0), NULL))
     AS Incasari,
     TO_NUMBER( SUBSTR ( TO_CHAR (
          SUM(DISTINCT 1000000000000000 *
          LF.NrFact+100000000000000 *
          LF.Linie+Cantitate * PretUnit * (1+ProcTVA)),
     '99999999999999999999999999'), 14,15)) -•
     SUM (DECODE (LF.Linie, 1, NVL(Transa,0), NULL))
     AS De_Incasat FROM PRODUSE P, LINIIFACT LF, FACTURI F,
     CLIENŢI C, INCASFACT I WHERE P.CodPr = LF.CodPr AND
LF.NrFact = F.NrFact AND F.CodCl = C.CodCl AND F.Nrfact=1.NrFact
(+)
GROUP BY DenCl HAVING
     TO_NUMBER( SUBSTR ( TO_CHAR (
       SUM(DISTINCT 1000000000000000 * LF.NrFact+100000000000000 *
         LF.Linie+Cantitate * PretUnit * (1+ProcTVA)),
         -99999999999999999999999999'),          14,15))   -
     SUM (DECODE (LF.Linie, 1, NVL(Transa,0), NULL))
>= ALL
     (SELECT TO_NUMBER( SUBSTR ( TO_CHAR (
       SUM(DISTINCT 1000000000000000 * LF.NrFact+100000000000000 *
              . LF.Linie+Cantitate * PretUnit * (1+ProcTVA)),
    '99999999999999999999999999'), 14,15)) - SUM (DECODE (LF.Linie, 1,
                          NVL(Transa,0), NULL))
     FROM PRODUSE P, LINIIFACT LF, FACTURI F,
          CLIENŢI C, INCASFACT I WHERE P.CodPr = LF.CodPr AND
     LF.NrFact = F.NrFact AND F.CodCl = C.CodCl AND F.Nrfact=I.NrFact
     (+)
     GROUP BY DenCl)
```

Din nou despre diviziunea relaţională

Revenind la cazul teoretic din capitolul 2 (figura 2.56), câtul diviziunii relaţionale a Rl: R2 (altfel spus, găsirea tuturor „icşilor" care sunt în Rl în combinaţie cu toţi „igrecii" din R2) se calculează în SQL şi astfel:

```
SELECT
X FROM
```

```
Rl
GROUP
BY X
HAVING COUNT(Y) = (SELECT COUNT(Y)
        FROM R2)
```

Căror clienţi li s-a trimis măcar o factură în toate zilele cu vânzări?

```
SELECT DenCl
FROM FACTURI F, CLIENŢI C WHERE F.CodCl=C.CodCI
GROUP BY DenCl
HAVING COUNT(DISTINCT DataFact) =
   (SELECT COUNT (DISTINCT DataFact)
   FROM FACTURI)
```

Modest ca volum (nu şi ca importanţă), răspunsul se găseşte în figura 5.43.

DENCL
Client 1 SRL

Figura 5.43. Clientul pentru care există cel puţin o factură în fiecare zi

Ce produse au fost vândute tuturor clienţilor?

```
SELECT DenPr
FROM PRODUSE P, LINIIFACT LF, FACTURI F WHERE
P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact GROUP BY
DenPr
HAVING COUNT(DISTINCT CodCI) =
   (SELECT COUNT (CodCI)
   FROM CLIENŢI)
```

Răspunsul este *Produs 2.*

5.7. Secvenţializarea interogărilor

Cu tot ataşamentul pentru Visual FoxPro, în materie de SQL (şi nu numai) există un decalaj serios faţă de standardul SQL-92 şi faţă de SGBD-urile profesionale: (IBM) DB2, Oracle, Informix, Sybase, (Microsoft) SQL Server etc.

Aceasta este vestea rea. Vestea bună e că mai toate interogările dificile, ce nu pot fi formulate printr-o singură frază SELECT principală (plus subconsultări pe un singur nivel), au leac în VFP salvând rezultatul unei interogări în tabele temporare (cursoare), tabele derivate sau chiar tabele obişnuite care devin „materie primă" pentru alte fraze SELECT ş.a.m.d. Prin dispunerea succesivă a interogărilor se redactează programe (. PRG) lansate în execuţie atunci când se doreşte informaţia respectivă sau când se obţine un raport.

Uneori însă, şi în SGBD-urile din lumea foarte bună este necesară salvarea rezultatelor intermediare în cadrul aceleiaşi interogări sau pentru prelucrări ulterioare. De obicei, tabelele derivate constituie un suport universal pentru asemenea gen de operaţiuni. Sunt însă şi soluţii mai la îndemână. Ceva mai înainte evocam cursoarele VFP. în DB2, tabelele temporare îmbracă forma expresiilor-tabele care sunt tabele temporare a căror definiţie este păstrată numai pe parcursul „vieţii" interogării în care este inclusă. Prin urmare, nu pot fi folosite pentru „pasarea" rezultatelor intermediare între SELECT-uri independente.

Fraze SELECT independente în VFP

Ne vom focaliza atenţia mai întâi pe VFP, pentru că aici avem de recuperat câteva handicapuri: un singur nivel de subconsultare, subconsultări în clauza HAVING şi subconsultări în FROM.

Să se afişeze, pentru fiecare factură, dacă este fără nici o încasare, încasată parţial sau încasată total (şi cele trei valori: facturată, încasată şi rămasă de încasat)?

Am ales pentru început una dintre primele probleme care creează necazuri în VFP în varianta SQL curată, adică neprocedurală. Listingul 5.2 conţine o soluţie mult mai simplă alcătuită din trei fraze SQL independente. Primele două obţin cursoarele utilizate în a treia. Cursorul este, cel puţin pentru exemplele noastre în VFP, o tabelă temporară: poate fi închisă explicit sau la ieşirea din VFP; la închidere se şterge, deci după utilizare nu mai ocupă spaţiu pe disc.

Listing 5.2. Situaţia încasării fiecărei facturi

```
* cursorul cVINZARI conţine valoarea fiecărei facturi
SELECT NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA)) ;
   AS Facturat ;
FROM LINIIFACT LF INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr ;
INTO CURSOR cVINZARI ;
GROUP BY NrFact
* cINCASARI conţine valoarea
* incasata pentru fiecare factura SELECT
NrFact, SUM(Transa) AS încasat ;
FROM INCASFACT ;
INTO CURSOR cINCASARI ;
GROUP BY NrFact
*   se            jonctioneaza               (extern) cele doua cursoare
SELECT V.NrFact, Facturat, NVL(încasat,0) AS încasat,      ;
   Facturat - NVL(încasat,0) AS Diferenţa, ;
   IIF(Facturat=NVL(încasat,0), 'ÎNCASATA TOTAL ', ;
      IIF(Facturat> NVL(încasat,0) AND NVL(încasat,0)    > 0,      ;
         'încasataparţial ', 'Fara nici o incasare'))      ;
   AS Situatiune ;
FROM cVINZARI V LEFT OUTER JOIN cINCASARI I ON V.NrFact=I.NrFact
```

Reproşul principal adus acestei soluţii VFP este... proceduraiitatea, în vădită contradicţie cu filosofia SQL — curat neprocedurală, frecând peste supărare, ajungem şi la principalul atu: se oferă, totuşi, răspunsul corect la problema formulată.

Care sunt clienţii pentru care există cel puţin câte o factură emisă în fiecare zi?

Varianta procedurală este prezentată în Iistingul 5.3.

Listing 5.3. Clienţii pentru care există cel puţin câte o factură emisă în fiecare zi

```
* Produsul cartezian al valorilor DenCl si DataFact SELECT DISTINCT
DenCl, DataFact ;
FROM CLIENŢI, FACTURI ;
INTO CPRSOR cPRODUS_CARTEZIAN

* Clienţii si zilele in care s-au emis facturi pentru ei SELECT
DISTINCT DenCl, DataFact ;
FROM CLIENŢI INNER JOIN FACTURI ON CLIENŢI.CodCl=FACTURI.CodCI ;
INTO CURSOR cRl

   Valorile DenCl care nu se regăsesc combinate cu toate valorile
DataFact
SELECT DISTINCT DenCl ;
FROM cProdus_Cartezian ;
INTO CURSOR cNu_I ;
```

```
WHERE DenCl+DTOC(DataFact) NOT IN ;
   (SELECT DenCl+DTOC(DataFact) ;
   FROM cRl)

* in fine, rezultatul SELECT DISTINCT DenCl ;
FROM cRl ;
WHERE DenCl NOT IN ;
   (SELECT DenCl ;
   FROM cNu_I)
```

Continuăm cu alte exemple care au fost, până acum, frunstrante pentru fox-işti, cele în care în clauza `HAVING` apare o subconsultare.

Care este ziua în care s-au emis cele mai multe facturi?

Practic, în prezentul caz, ca şi în celelalte de acest tip, prin cursoare, grupurile sunt transformate în tupluri şi, în loc de `HAVING`, se va folosi `WHERE`.

Listing 5.4. Ziua (zilele) în care s-au emis cele mai multe facturi?

```
SELECT DataFact, COUNT(*) AS Nr ;
FROM FACTURI ;
INTO CURSOR cNrFact_Zilnic ;
GROUP BY DataFact –
SELECT DataFact, Nr ;
FROM cNrFact_Zilnic ;
WHERE Nr IN ;
   (SELECT MAX(Nr) ;
   FROM cNrFact_Zilnic)
```

Care este judeţul în care berea s-a vândut cel mai bine?

După cum se observă, am selectat numai exemplele de importanţă majoră pentru ţară. Răspunsul la această tulburătoare problemă este oferit de programul din listing 5.5.

Listing 5.5. Judeţul în care berea s-a vândut cel mai bine

```
SELECT Judeţ, SUM(Cantitate * PretUnit * (1+ProcTVA)) ;
   AS Vinzari_Bere ;
FROM JUDEŢE J, LOCALITATI L, CLIENŢI C, FACTURI F, ;
   LINIIFACT LF, PRODUSE P ;
INTO CURSOR cBere ;
WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND C.CodCl=F.CodCl ;
AND F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr AND Grupa='Bere' ;
GROUP BY Judeţ

SELECT Judeţ, Vinzari_Bere ;
FROM cBere ;
WHERE Vinzari_Bere >= ALL ;
      (SELECT DISTINCT Vinzari__Bere ;
      FROM cBere)
```

Care este judeţul cu vânzări imediat superioare judeţului Neamţ?

Soluţia din listingul 5.6 are un dram suplimentar de interes, deoarece la ultima sa interogare, în afara clauzei WHERE care conţine două subconsultări, se efectuează theta-joncţiunea dintre cele două cursoare, cVinzari_Neamt şi cVinzari_Judete.

Listing 5.6. Judeţul cu vânzări imediat peste cele ale judeţului Neamţ ,

```
* Vinzarile jud. Neamţ
SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari ;
FROM JUDEŢE J ;
```

```
   INNER JOIN LOCALITATI L ON J.Jud=L.Jud ;
   INNER JOIN CLIENŢI C ON L.CodPost=C.CodPost ;
   INNER JOIN FACTURI F ON C.CodCl=F.CodCl ;
   INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
   INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
INTO CURSOR cVinzari_Neamt ;
WHERE Judet='Neamţ'

* Vinzarile fiecărui judeţ
SELECT Judeţ, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari ;
FROM JUDEŢE J ;
   INNER JOIN LOCALITATI L ON J.Jud=L.Jud ;
   INNER JOIN CLIENŢI C ON L.CodPost=C.CodPost ;
   INNER JOIN FACTURI F ON C.CodCl=F.CodCl ;
   INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
   INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
INTO CURSOR cVinzari_Judete ;
GROUP BY Judeţ

SELECT Judeţ, Vinzari ;
FROM cVinzari_Judete ;
WHERE Vinzari <= ALL ;
   (SELECT cVinzari_Judete.Vinzari ;
   FROM cVinzari Judeţe, cVinzari Neamţ ;
   WHERE cVinzari_Judete.Vinzari > cVinzari_Neamt.Vinzari) ;
   AND Vinzari > ALL ;
   . (SELECT Vinzari ;
      FROM cVinzari_Neamt)
```

Care este clientul cel mai mare datornic?

Este de-a dreptul reconfortant că nu trebuie să mai apelăm la artificiile pentru calculul valorilor facturate şi încasate - vezi listingul 5.7.

Listing 5.7. Clientul cu cel mai mare rest de plată

```
* Valorile facturate, pe clienţi
SELECT CodCI, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat ;
FROM FACTURI F ;
   INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
   INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
INTO CURSOR cFACTURAT  ;
GROUP BY CodCI
* Valorile incasate, pe clienţi
SELECT CodCI, SUM(Transa) AS încasat
;
FROM FACTURI F ;
   INNER JOIN INCASFACT I ON F.NrFact=I.NrFact ;
INTO CURSOR cINCASAT ;
GROUP BY CodCI

* Restul de plata, pe clienţi
SELECT cFACTURAT.CodCI, Facturat, NVL(încasat,0) AS încasat, ;
   Facturat - NVL(Incasat,0) AS De_Incasat ;
FROM cFACTURAT LEFT OUTER JOIN cINCASAT ;
   ON cFACTURAT.CodCl=cINCASAT.CodCI ;
INTO CURSOR cDe_INCASAT ;

* Finalul apoteotic
SELECT DenCl, cDe_INCASAT.* ;
FROM cDe_INCASAT INNER JOIN CLIENŢI ;
   ON cDe_INCASAT.CodCl=CLIENTI.CodCI ;
WHERE De_Incasat >= ALL ;
```

```
        (SELECT De_Incasat ;
        FROM cDe_INCASAT)
```

Căror clienţi li s-a trimis măcar o factură în toate zilele de vânzare?

Nici problemele ce impun soluţii de tip diviziune relaţională nu mai constituie o dificultate urmând logica interogărilor independente - listingul 5.8.

Listing 5.8. Clientul căruia i s-a trimis măcar o factură în toate zilele de vânzare

```
SELECT CodCI, COUNT(DISTINCT DataFact) AS Cite_Zile ;
FROM FACTURI ;
INTO CURSOR cl ;
GROUP BY CodCI
SELECT DenCl ;
FROM CLIENŢI INNER JOIN cl ON CLIENŢI.CodCl=cl.CodCI ;
WHERE Cite_Zile =ANY ;
   (SELECT COUNT(DISTINCT DataFact) ;
   FROM FACTURI)
```

Scripturi cu interogări secvenţiale în Oracle şi DB2

Deşi oarecum stânjenitoare, secvenţializarea interogărilor se poartă şi la case mai mari. Una dintre ele - Oracle, iar alta IBM DB2. Fie că este vorba despre cunoştinţe nu prea avansate de SQL, fie despre o anumită comoditate, cert este că de multe ori interogări insolubile se pot rezolva ceva mai primitiv.

Care este clientul cu cel mai mare rest de plată?

încercăm să facem uitată penibila soluţie formulată în paragraful precent, ce-i drept, înlocuind-o cu o soluţie tot din „lumea a treia" a SQL-ului, deoarece se bazează pe un script în care rezultatele intermediare ale consultărilor se salvează nu în cursoare, ci în tabele derivate.

Listing 5.9. Script Oracle de aflare a clientului cu cel mai mare rest de plată

```
— Valorile facturate, pe clienţi
DROP VIEW vDe_Incasat ;
DROP VIEW vlncasat ;
DROP VIEW vFacturat ;
CREATE VIEW vFacturat AS
   SELECT CodCl, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat
   FROM FACTURI F, LINIIFACT LF, PRODUSE P WHERE F.NrFact=LF.NrFact AND
   LF.CodPr=P.CodPr GROUP BY CodCl ;
-- Valorile incasate, pe clienţi
CREATE VIEW vlncasat AS
   SELECT CodCl, SUM(Transa) AS încasat
   FROM FACTURI F, INCASFACT I WHERE
   F.NrFact=I.NrFact GROUP BY CodCl ;
— Restul de plata, pe clienţi
CREATE VIEW vDe_INCASAT AS
   SELECT VFACTURAT.CodCl, Facturat, NVL(încasat,0) AS încasat,
   Facturat - NVL(încasat,0) AS De_Incasat FROM VFACTURAT,
   vINCASAT
   WHERE VFACTURAT.CodCl=vINCASAT.CodCl (+)    ;
— Finalul
SELECT DenCl, vDe__INCASAT. *
FROM vDe_INCASAT, CLIENŢI WHERE
vDe_INCASAT.CodCl=CLIENTI.CodCl AND
De_Incasat >= ALL
           (SELECT De_Incasat FROM
           vDe_INCASAT)      ;
```

De remarcat că şi în VFP, în loc de cursoare puteam folosi tabele derivate...

Care sunt primii trei clienţi în ordinea descrescătoare a datoriilor (cei mai mari trei datornici)?

Iată o interogare al cărei răspuns ar interesa multă lume, mai ales dacă baza de date ar fi la nivel naţional... Cu toate opintelile de care suntem în stare, la acest moment ne este cu neputinţă să răspundem altfel decât printr-un script - listingul 5.10. Din nefericire, soluţia a fost îngreunată de faptul că nu se poate folosi clauza ORDER BY într-o comandă CREATE VIEW.

Listing 5.10. Script Oracle pentru extragerea primilor 3 datornici

```
DROP VIEW vOrdonare ;
DROP VIEW vDe_Incasat ;
DROP VIEW vIncasat ;
DROP VIEW vFacturat ;
—  Valorile facturate, pe clienţi CREATE VIEW vFacturat AS
   SELECT CodCI, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat
   FROM FACTURI F, LINIIFACT LF, PRODUSE P WHERE F.NrFact=LF.NrFact AND
   LF.CodPr=P.CodPr GROUP BY CodCI WITH READ ONLY;
—  Valorile incasate, pe clienţi CREATE VIEW vIncasat AS
   SELECT CodCI, SUM(Transa) AS încasat
   FROM FACTURI F, INCASFACT I
   WHERE F.NrFact=I.NrFact
   GROUP BY CodCI
   WITH READ ONLY ;
-- Restul de plata, pe clienţi, in ordinea
—  (descrescătoare a) valorii CREATE VIEW vDe_INCASAT AS
   SELECT DenCl, vFACTURAT.CodCI, Facturat, NVL(încasat,0) AS încasat,
     Facturat - NVL(încasat,0) AS De^Incasat FROM
   VFACTURAT, vINCASAT, CLIENŢI WHERE
   VFACTURAT.CodCl=CLIENTI.CodCI
     AND VFACTURAT.CodCl=vINCASAT.CodCI (+)
WITH READ ONLY ;
—  Primii trei datornici CREATE VIEW vORDONARE AS
   SELECT *
                  FROM vDe_INCASAT WHERE De_Incasat >=
     (SELECT MAX(De_Incasat)
     FROM vDe_INCASAT WHERE De_Incasat <
       (SELECT MAX(De_Incasat)
       FROM vDe_INCASAT WHERE De_Incasat <
          (SELECT MAX(De_Incasat)
          FROM vDe_INCASAT)
     )
     )
WITH READ ONLY ;
—  Ordonare finala SELECT *
FROM vOrdonare
ORDER BY De încasat DESC
```

DENCL	CODCL	FACTURAT	INCASAT	DEJNCASAT
Client 3 SRL	1003	7242935	0	7242935
Client 6 SA	1006	6786570	0	6786570
Client 4	1004	5438300	0	5438300

Figura 5.44. Primii trei datornici

Expresii-tabele în DB2

Varianta de rezolvare a problemei formulate mai sus funcţionează aproape identic şi în DB2. secvenţa de comenzi din listingul 5.11 producând un rezultat identic celui din figura 5.43.

Listing 5.11. Script DB2 pentru extragerea primilor 3 datornici

```
DROP VIEW vOrdonare ;
DROP VIEW vDe_Incasat ;
DROP VIEW vIncasat ;
```

Rezultatul este prezentat în figura 5.44.

```
DROP VIEW vFacturat ;
—  Valorile facturate, pe clienţi CREATE VIEW vFacturat AS
   SELECT CodCI, SUM(Cantitate * PretUnit * (1+ProcTVA))
      AS Facturat FROM FACTURI F, LINIIFACT LF,
   PRODUSE P WHERE F.NrFact=LF.NrFact AND
   LF.CodPr=P.CodPr GROUP BY CodCI ;
—  Valorile incasate, pe clienţi CREATE VIEW vIncasat AS
   SELECT CodCI, SUM(Transa) AS încasat FROM FACTURI F,
   INCASFACT I WHERE F.NrFact=I.NrFact GROUP BY CodCI ;
—  Restul de plata, pe clienţi, in ordinea (descrescătoare a)
valorii
CREATE VIEW vDe_INCASAT AS
   SELECT DenCl, VFACTURAT.CodCI, Facturat, VALUE(încasat,0)
      AS încasat,
   Facturat - VALUE(încasat,0) AS De_Incasat FROM VFACTURAT
   INNER JOIN CLIENŢI ON vFacturat.CodCl=CLIENTI.CodCI LEFT
   OUTER JOIN vINCASAT ON vFACTURAT.CodCl=vINCASAT.CodCI ;
—  Primii trei datornici CREATE VIEW vORDONARE AS
   SELECT *
   FROM vDe_INCASAT WHERE De_Incasat >=
      (SELECT MAX(De_Incasat)
      FROM vDe_INCASAT WHERE De încasat <
```

```
        (SELECT MAX(De_Incasat)
        FROM vDe_INCASAT WHERE
        De_Incasat <
           (SELECT MAX(De_Incasat)
           FROM vDe_INCASAT)
     )
  ) ;
-- Ordonare finala SELECT *
FROM vOrdonare
ORDER BY De_Incasat DESC ;
```

DB2 prezintă însă şi un ingredient interesant de rezolvare a interogărilor mai complexe, *expresiile-tabele,* care sunt apropiate de „filosofia" cursoarelor din VFP, deşi, spre deosebire de acestea, nu pot fi închise şi deschise explicit, iar „viaţa" lor se reduce numai la interogarea care le utilizează.

Spre deosebire de scripturile prezentate în acest paragraf, interogările bazate pe expresii-tabele sunt privite cu mult mai multă indulgenţă în comunitatea SQL, aceasta deoarece soluţia se reduce la o singură frază SELECT principală, precedată de consultările pentru definirea expresiilor-tabele, subconsultări care se consideră a face parte integrantă din interogare.

Revenim (pentru a treia oară) la exemplul 19 din algebra relaţională: *Ce facturi au fost emise în aceeaşi zi cu factura 1120?*

Iată şi soluţia DB2 bazată pe expresii-tabele:

```
WITH
Zi_1120AS
   (SELECT DataFact
   FROM FACTURI
   WHERE NrFact=1120)
SELECT NrFact
FROM FACTURI, Zi_1120
WHERE FACTURI. DataFact=Zi__l 12 0 . DataFact
```

ZU120 este o expresie-tabelă cu o singură coloană — DataFact — şi o singură linie care conţine data întocmirii facturii 1120, fiind folosită în fraza SELECT principală ca o tabelă obişnuită.

In ce zile s-au vândut şi produsul cu denumirea „ Produs 1", şi cel cu denumirea „ Produs 2"? (exemplul 17 - algebra relaţională)

```
   WITH
   ZILE_PRODU S1 AS
     (SELECT DISTINCT DataFact FROM PRODUSE,
     LINIIFACT, FACTURI WHERE PRODUSE.CodPr =
     LINIIFACT.CodPr AND LINIIFACT.Nrfact =
     FACTURI.NrFact AND DenPr = 'Produs 1'),
   ZILE_PRODUS2 AS
     (SELECT DISTINCT DataFact FROM
     PRODUSE, LINIIFACT, FACTURI
  WHERE  PRODUSE.CodPr  =  LINIIFACT.CodPr  AND
      LINIIFACT.Nrfact   =   FACTURI.NrFact   AND
      DenPr = 'Produs 2')
SELECT *
FROM ZILE_PRODUSl INTERSECT SELECT *
FROM ZILE_PRODUS 2
```

De această dată au fost definite două expresii-tabele, ZILE_PRODUSl şi ZILE_PRODUS2. Prima conţine zilele în care s-a vândut produsul 1, iar a doua - zilele în care s-a vândut cel de-al doilea produs. Fraza SELECT principală îndeplineşte o simplă formalitate - intersecţia celor două tabele ad-hoc.

Care sunt clienţii pentru care există cel puţin câte o factură emisă în fiecare zi? (exemplul 22 din algebra relaţională - figura 2.27)

Şi diviziunea relaţională se simplifică serios în noile condiţii.

```
WITH
CLIENTI_NRZILE AS
   (SELECT DenCl, COUNT(DISTINCT DataFact) AS NrZile FROM CLIENŢI C
   INNER JOIN FACTURI F ON C.CodCl=F.CodCI GROUP BY DenCl),
NRZILE AS
   (SELECT COUNT(DISTINCT DataFact) AS NrZile FROM FACTURI)
SELECT DenCl, NrZile FROM CLIENTI_NRZILE
WHERE NrZile IN (SELECT NrZile FROM NRZILE)
```

După cum se observă, expresiile-tabelă pot fi obţinute şi prin consultări ce conţin GROUP BY,. iar pe de altă parte, în fraza SELECT principală pot fi formulate subconsultări în care apar, pe orice nivel, atribute din expresiile-tabele.

Care este judeţul în care berea s-a vândut cel mai bine?

```
WITH
JUDETE_BERE1 AS
   (SELECT Judeţ, SUM(Cantitate * PretUnit * (1+ProcTVA))
      AS Vinzari_Bere FROM JUDEŢE J, LOCALITATI L,
   CLIENŢI C, FACTURI F,
      LINIIFACT LF, PRODUSE P WHERE J.Jud=L.Jud AND
   L.CodPost=C.CodPost AND C.CodCl=F.CodCI AND
   F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr AND
   Grupa='Bere'
   GROUP BY Judeţ),
BERE_JUDEŢE AS
   (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
      AS Vinzari Bere
   FROM JUDEŢE J, LOCALITATI L, CLIENŢI C, FACTURI F,
     LINIIFACT LF, PRODUSE P WHERE J.Jud=L.Jud
   AND L.CodPost=C.CodPost AND C.CodCl=F.CodCl AND
   F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr AND
   Grupa='Bere'
   GROUP BY Judeţ)
SELECT Judeţ, Vinzari_Bere
FROM JUDETE_BERE1
WHERE Vinzari_Bere >= ALL (SELECT Vinzari_Bere FROM BERE_JUDEŢE)
```

Care este clientul cu cel mai mare rest de plată?

Revenim la una dintre cele mai dificile probleme de până acum, de data aceasta cu o soluţie „apetisantă".

```
WITH
FACTURAT AS
   (SELECT CodCl, SUM(Cantitate * PretUnit * (1+ProcTVA))
     AS Facturat FROM FACTURI F, LINIIFACT LF,
   PRODUSE P WHERE F.NrFact=LF.NrFact AND
```

```
   LF.CodPr=P.CodPr GROUP BY CodCl),
ÎNCASAT AS
   (SELECT CodCl, SUM(Transa) AS încasat FROM FACTURI F,
   INCASFACT I WHERE F.NrFact=I.NrFact GROUP BY CodCl)
SELECT DenCl, Facturat, VALUE(încasat,0) AS încasat,
   Facturat - VALUE(încasat,0) AS De_Incasat
FROM CLIENŢI INNER JOIN FACTURAT ON CLIENŢI.CodCl=FACTURAT.CodCl
   LEFT OUTER JOIN ÎNCASAT ON FACTURAT . CodCl-'rvCASAT. CodCl
 WHERE Facturat - VALUE(încasat,0) >= ALL
      (SELECT Facturat - VALUE(încasat,0)
    FROM FACTURAT LEFT OUTER JOIN ÎNCASAT
         ON FACTURAT.CodCl=INCASAT.CodCl)
```

Capitolul 6 SQL ŞI MAI AVANSAT

Primul dintre capitolele dedicate frazelor SELECT, 4, se numeşte *Elemente de bază ale interogărilor SQL,* iar capitolul 5 - *SQL (ceva) mai avansat.* Cum prezentul capitol a fost intitulat *SQL şi mai avansat,* în mod normal, dacă şi următorul continua problematica interogărilor, ar fi trebuit să se numescă *SQL al naibii de avansat.* Din fericire, acesta este ultimul în care se discută despre opţiuni de consultare, câteva opţiuni care se încadrează în elementele de fineţe ale limbajului. Concret, vor fi abordate: corelarea simplă şi dublă cu şi fară ajutorul operatorului EXISTS, subinterogări în clauza FROM, interogările scalare şi cele ierarhice.

6.1. Interogări corelate. Operatorul EXISTS

Ca şi IN, operatorul EXISTS permite legarea frazei SELECT principale de una sau mai multe subconsultări şi, astfel, formularea unor interogări complexe. Prin EXISTS se defineşte un predicat care are valoarea logică „adevărat" dacă subconsultarea s-a concretizat într-o tabelă ce are cel puţin o linie.

în general, EXISTS poate înlocui cu succes operatorul IN. Reciproca nefiind valabilă'[4], unii autori, precum C.J. Date, îl preferă şi recomandă la formularea subconsultărilor. Pe de altă parte însă, EXISTS este unul dintre cei mai dificili operatori. Logica sa se deosebeşte radical de cea a lui IN.

Utilizând operatorul IN, în tabela-rezultat se extrag acele linii din tabela (tabelele) din fraza SELECT principală care satisfac o condiţie aplicată liniilor din subconsultare. La utilizarea lui EXISTS se extrag linii tot ale tabelei/tabelelor frazei SELECT principală, dar pe baza existenţei a cel puţin o linie în tabela/tabelele subconsultării, care satisface (satisfac),' pe lângă restricţii proprii, şi condiţii formulate luându-se în considerare atribute/tabele din fraza principală (cam întortocheat, nu-i aşa?).

Predicatele din clauza WHERE operează la nivel de linie, şi nu la nivel de ansamblu (seturi de înregistrări), ceea ce, după Joe Celko, este o deviere de la spiritul relaţional. Pe de altă parte, Sheryl Larsen[29] pledează pentru folosirea lui EXISTS în detrimentul IN-ului pe considerentul optimizării. IN presupune crearea unor tabele temporare ce reclamă uneori timp de execuţie sensibil mai mare prin comparaţie cu logica tuplu-cu-tuplu EXISTS-enţială.

29 [Larsen96],

Pentru comparaţie, începem cu exemple formulate în capitolul precedent la prezentarea operatorului IN.

Ce facturi au fost emise în aceeaşi zi cu factura 1120?

Varianta de interogare bazată pe operatorul EXISTS se prezintă astfel:

```
SELECT      NrFact
FROM FACTURI FI
WHERE EXISTS
        (SELECT *
        FROM FACTURI F2
        WHERE F2.NrFact=1120 AND FI.DataFact=F2.DataFact)
```

Logica execuţiei se derulează după un scenariu *top-bottom-top* (sus-jos-sus) prezentat în figura 6.1.

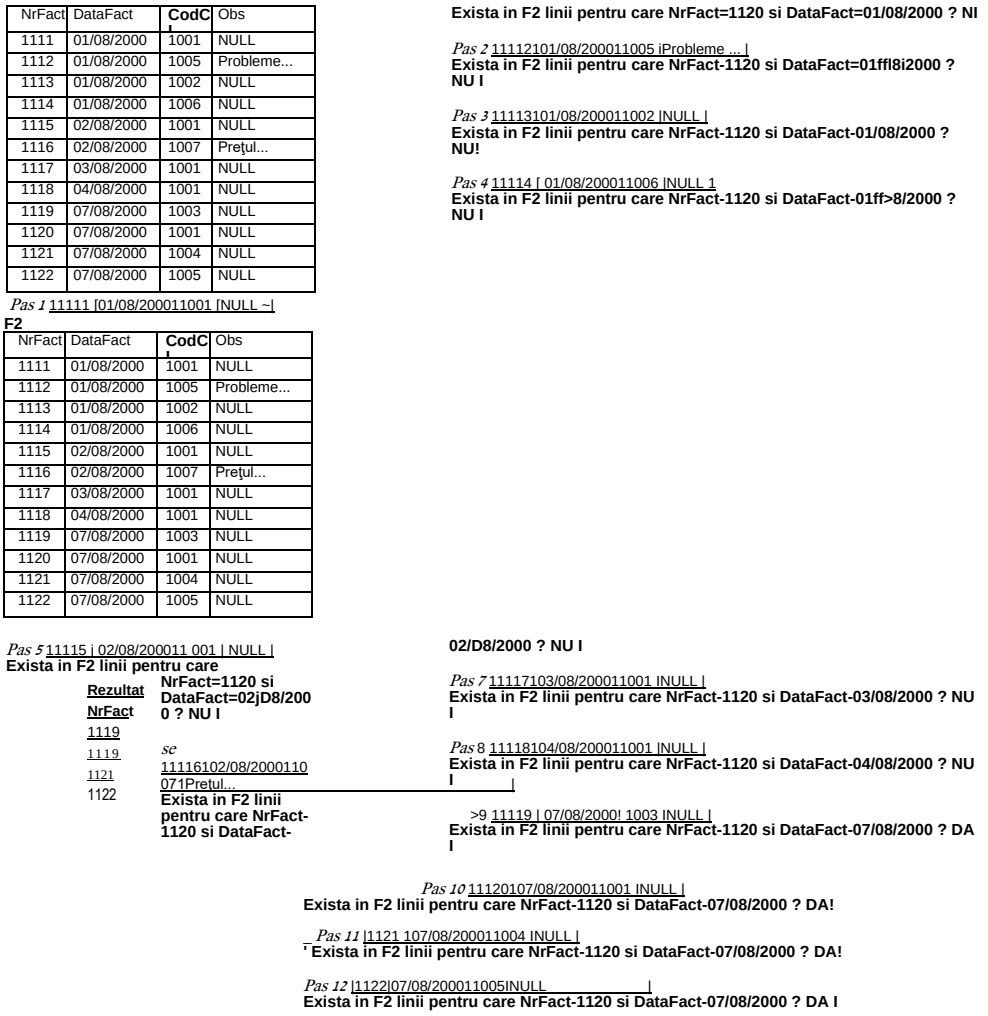

NrFact DataFact CodCt Obs
1111 01/08/2000 1001 NULL
1112 01/08/2000 1005 Probleme...
1113 01/08/2000 1002 NULL
1114 01/08/2000 1006 NULL
1115 02/08/2000 1001 NULL
1116 02/08/2000 1007 Preţul...
1117 03/08/2000 1001 NULL
1118 04/08/2000 1001 NULL
1119 07/08/2000 1003 NULL
1120 07/08/2000 1001 NULL
1121 07/08/2000 1004 NULL
1122 07/08/2000 1005 NULL
Exista in F2 linii pentru care NrFact=1120 si DataFact=01/08/2000 ? NI
Pas 1 11111 [01/08/200011001 [NULL ~|
F2
Pas 2 11112101/08/200011005 iProbleme ... |
Exista in F2 linii pentru care NrFact-1120 si DataFact=01ffl8i2000 ? NU I
Pas 3 11113101/08/200011002 |NULL |
Exista in F2 linii pentru care NrFact-1120 si DataFact-01/08/2000 ? NU!
Pas 4 11114 [01/08/200011006 |NULL 1
Exista in F2 linii pentru care NrFact-1120 si DataFact-01ff>8/2000 ? NU I
Pas 5 11115 i 02/08/200011 001 | NULL |
Exista in F2 linii pentru care
Rezultat
NrFact
1119
1119
1121
1122
NrFact=1120 si DataFact=02jD8/2000 ? NU I
se
11116102/08/2000110071Preţul...
Exista in F2 linii pentru care NrFact-1120 si DataFact-
02/D8/2000 ? NU I
Pas 7 11117103/08/200011001 |NULL |
Exista in F2 linii pentru care NrFact-1120 si DataFact-03/08/2000 ? NU I
Pas 8 11118104/08/200011001 |NULL |
Exista in F2 linii pentru care NrFact-1120 si DataFact-04/08/2000 ? NU I
>9 11119 | 07/08/2000! 1003 |NULL |
Exista in F2 linii pentru care NrFact-1120 si DataFact-07/08/2000 ? DA I
Pas 10 11120107/08/200011001 |NULL |
Exista in F2 linii pentru care NrFact-1120 si DataFact-07/08/2000 ? DA!
Pas 11 |1121 107/08/200011004 |NULL |
Exista in F2 linii pentru care NrFact-1120 si DataFact-07/08/2000 ? DA!
Pas 12 |1122|07/08/200011005|NULL |
Exista in F2 linii pentru care NrFact-1120 si DataFact-07/08/2000 ? DA I

Figura 6.1. Logica operatorului EXISTS

Elementul de dificultate ţine de modalitatea de exprimare a condiţiei, care este una indirectă. Se parcurge fiecare linie din SELECT-ul principal, examinându-se, pe rând, dacă există măcar o linie în subconsultare care să satisfacă predicatul subconsultării. Dacă da, atunci nu aceasta se inserează în rezultat, ci linia din SELECT-ul principal! Nu ştiu cum e mai nimerit să-i zicem, lucru în echipă sau însuşirea de către SELECT-ul principal a meritelor SELECT-ului „subaltern”.

Dacă se doreşte eliminarea din rezultat şi a facturii de referinţă, 1120, se modifică uşor interogarea:

```
SELECT NrFact FROM FACTURI' FI
WHERE NrFact <> 1120 AND
EXISTS
    (SELECT DataFact FROM FACTURI F2
    WHERE F2.NrFact=1120 AND FI.DataFact=F2.DataFact)
```

Interesant este că interogările corelate pot fi legate nu numai prin operatorul EXISTS, ci şi de IN. Altfel spus, interogarea următoare este perfect similară anterioarei:

```
SELECT NrFact FROM FACTURI FI
WHERE NrFact <> 1120 AND
DataFact IN
    (SELECT DataFact FROM FACTURI F2
    WHERE NrFact=1120 AND FI.DataFact=F2.DataFact)
```

Ce facturi au fost emise în alte zile decât factura 1120?
Următoarele două soluţii conduc la acelaşi rezultat:

```
    SELECT NrFact FROM FACTURI FI WHERE NOT EXISTS (SELECT *
    FROM FACTURI F2
    WHERE F2.NrFact=112 0 AND FI.DataFact=F2.DataFact)
```

sau

```
    SELECT NrFact FROM FACTURI FI WHERE EXISTS (SELECT *
    FROM FACTURI F2
    WHERE F2.NrFact=1120 AND FI.DataFact<>F2.DataFact)
```

în subconsultarea corelată (cea care succedă operatorului EXISTS), clauza SELECT poate conţine ca argument * sau un atribut care, de preferinţă, nu are valori nule. Cea mai simpli sigură şi rapidă (pentru SBGD) variantă este însă folosirea unei constante. Spre exemplu, ultim; variantă se poate schimba, aproape insesizabil, în:

```
SELECT NrFact FROM FACTURI FI WHERE EXISTS (SELECT 1 FROM FACTURI
                                            F2
    WHERE F2 .NrFact=1120 AND FI. DataFactOF2 . DataFact)
```

înlocuirea asteriscului sau a oricărui alt atribut cu 1 nu modifică în nici un fel corectitudinea interogării şi este de bun augur pentru simplitate şi viteza de execuţie.
Care sunt clienţii cărora li s-au trimis facturi în aceeaşi zi în care a fost întocmită factura 1120?

- Soluţie SQL-92/DB2/Oracle:

```
SELECT DenCl FROM CLIENŢI CI WHERE EXISTS (SELECT 1 FROM FACTURI FI
                    WHERE FI.CodCl=Cl.CodCl AND EXISTS
          (SELECT 1          .
          FROM FACTURI F2 WHERE
          F2.NrFact=1120 AND
          FI.DataFact=F2.DataFact)
   )
```

- Soluţie VFP:

```
SELECT DenCl ;
FROM CLIENŢI C INNER JOIN FACTURI FI ;
   ON C.CodCl=Fl.CodCl ;                          .
WHERE EXISTS ;
      (SELECT 1 ;                    .
      FROM FACTURI F2 ;
      WHERE F2.NrFact=1120 AND FI.DataFact=F2.DataFact)
```

în varianta SQL-92/DB2/Oracle există două niveluri de subconsultare, însă corelarea este simplă, fiecare „etaj” fiind corelat numai la nivelul imediat superior. Peste numai câteva pagini vom discuta

câteva aspecte privind dubla corelare.

In ce judeţe s-a vândut produsul „ Produs 2 "?

Formulăm numai soluţia SQL-92/DB2/Oracle:

```
SELECT Judeţ FROM JUDEŢE J WHERE
EXISTS (SELECT 1 FROM LOCALITATI L
WHERE L.Jud=J.Jud AND EXISTS
(SELECT 1 FROM CLIENŢI C
          WHERE C.CodPost=L.CodPost AND EXISTS
             (SELECT 1 FROM FACTURI F
             WHERE F.CodCl=C.CodCl AND EXISTS
                (SELECT 1
                FROM LINIIFACT LF
                WHERE LF.NrFact=F.NrFact AND EXISTS
                   (SELECT 1 FROM PRODUSE P WHERE
                   P.CodPr=LF.CodPr AND
                      DenPr = 'Produs 2'
                   )
                )
             )
          )
   )
```

Această soluţie este una dintre cele mai bune, ca volum de resurse consumat, înlocuind atât costisitoarea joncţiune din interogările de tip INNER JOIN, cât şi numeroasele tabele intermediare presupuse de operatorul IN. Nu ne încearcă decât un regret: că nu funcţionează în VFP (din cauza multiplelor niveluri de consultare).

Prin EXISTS, firește, se poate realiza şi *intersecţia* a două relaţii. Dacă luăm în discuţie relaţiile Rl şi R2 (capitolul 2 - figura 2.4), intersecţia acestora se realizează şi astfel:

```
SELECT *
FROM Rl WHERE
EXISTS (SELECT 1
FROM R2
    WHERE Rl.A=R2.C AND Rl.B=R2.D AND R1.C=R2.E)
```

La prima vedere, nimic spectaculos. La a doua, descoperim că, spre deosebire de soluţia bazată pe IN, această frază funcţionează identic şi în VFP.

In ce zile s-au vândut şi produsul cu denumirea „ Produs I", şi cel cu denumirea „ Produs 2 ’’?

```
SELECT DISTINCT DataFact
FROM PRODUSE PI, LINIIFACT LF1, FACTURI FI WHERE PI.CodPr
= LF1.CodPr AND
    LF1.Nrfact = Fl.NrFact AND DenPr = 'Produs 1'
    AND EXISTS (SELECT 1
        FROM PRODUSE P2, LINIIFACT LF2, FACTURI F2 WHERE P2.CodPr = LF2.CodPr AND
    LF2.Nrfact = F2.NrFact AND P2.DenPr = 'Produs 2' AND F2.DataFact=Fl.DataFact)
```

Sunt corelate două instanţe ale joncţiunii PRODUSE-LINIIFACT-FACTURI; prima conţine liniile legate de *Produs I,* iar a doua de *Produs 2.* întrucât interesează zilele în care s-au vândut simultan cele două produse, atributul de corelare este Dat a Fact.

Ce clienţi au cumpărat şi „ Produs 2 ”, şi „ Produs 3 ”, dar nu au cumpărat „ Produs 5 ”?

```
SELECT DISTINCT DenCl
          FROM PRODUSE, LINIIFACT, FACTURI, CLIENŢI WHERE
      PRODUSE.CodPr = LINIIFACT.CodPr AND LINIIFACT.Nrfact =
```

```
                                        FACTURI.NrFact AND
    FACTURI.CodCI = CLIENŢI.CodCI AND DenPr = 'Produs 2' AND EXISTS (SELECT 1
    FROM PRODUSE, LINIIFACT, FACTURI
    WHERE PRODUSE.CodPr = LINIIFACT.CodPr AND
        LINIIFACT.NrFact = FACTURI.NrFact AND ' DenPr = 'Produs 3' AND
        CodCl=CLIENTI.CodCl)
                                AND NOT EXISTS (SELECT 1
        FROM PRODUSE, LINIIFACT, FACTURI WHERE PRODUSE.CodPr =
        LINIIFACT.CodPr AND LINIIFACT.Nrfact = FACTURI.NrFact AND
        DenPr = 'Produs 5' AND CodCl=CLIENTI.CodCl)
```

Fraza SELECT principală este corelată la două subconsultări dispuse „în linie", secvenţial. S-au folosit şi EXISTS şi NOT EXISTS. Nici aceasta nu este o dublă corelare, deoarece avem de-a face cu un singur nivel de subconsultare.

Care sunt clienţii cărora li s-au întocmit numai două facturi?

Este o problemă banală, pentru care se pot formula soluţii interesante. Fireşte, cea mai la îndemână variantă este:

```
SELECT      DenCl
    FROM
    CLIENŢI
    WHERE   CodCl
    IN      (SELECT
    CodCl   FROM
    FACTURI
    GROUP       BY
    CodCl HAVING
    COUNT(NrFact)
    = 2)
```

Şi mai simplă decât aceasta este următoarea:

```
SELECT      DenCl
FROM CLIENŢI
    INNER JOIN FACTURI ON CLIENŢI.CodCl=FACTURI.CodCl GROUP BY
DenCl HAVING COUNT(NrFact) = 2
```

Varianta pe care o consider cea mai interesantă nu foloseşte, pentru corelarea subconsultării cu fraza SELECT principală, nici EXISTS, nici IN:

```
SELECT      DenCl
FROM      CLIENŢI
WHERE 2 =
                                    (SELECT COUNT(*)
                                      FROM FACTURI
    WHERE FACTURI.CodCl=CLIENTI.CodCl)
```

Logica interogării:

Faza 1:

- **Se „încarcă" prima linie din CLIENŢI: (1001, 'Client 1 SRL', 'RIOOI', 'Tranziţiei, 13 bis', '6600', NULL).**
- **Se numără, prin funcţia** COUNT, **câte linii din FACTURI au** CodCl = 1001. **Rezultatul este 5.**
- **Se compară 2 din** SELECT-ul **principal cu rezultatul subconsultării corelate; 5 * 2, deci 'CLIENŢI' nu se include în rezultat**

Faza 2:

- Se „încarcă" a doua linie din CLIENŢI: (1002, 'Client 2 SA', 'R1002', NULL, '6600', '032-212121').
- Se numără, prin funcţia COUNT, câte linii din FACTURI au CodCI = 1002. Rezultatul este 1.
- 1 ^ 2, deci nici 'CLIENT2' nu se include în rezultat.

Faza 5:

« Se „încarcă" a cincea linie din CLIENŢI: (1005, 'Client 5 SRL', 'R1005', NULL, '1900', '056-111111').

e Se numără, prin funcţia COUNT, câte linii din FACTURI au CodCI = 1005.
Rezultatul este 2.

® 2 = 2, iar 'CLIENT5' se include în rezultat.

Din interogările menite să ilustreze folosirea operatorului EXISTS nu puteau lipsi cele legate de diviziunea relaţională. Reluăm exemplul simplu ce utilizează relaţiile Rl şi R2. Pentru a afla *valorile lui* X *aflate în Rl în combinaţie cu toate valorile lui* Y *din R2,* se poate recurge şi la soluţia SQL-92/DB2/Oracle:

```
SELECT DISTINCT
   X FROM Rl Tl_l
   WHERE EXISTS
   (SELECT 1
   FROM Rl Tl_2
   WHERE Tl__2
   .X = Tl_ 1 .X
   GROUP BY
   Tl_2.X
   HAVING COUNT(DISTINCT T1_2.Y) =
         (SELECT COUNT(Y)
         FROM R2)
   )
```

Se parcurge, pe rând, fiecare linie din Rl. De fiecare dată se verifică dacă, pentru X-ul curent, în Rl apar toţi igrecii din R2, verificare realizată prin GROUP BY şi HAVING. Pe baza acestei soluţii, se poate formula o altă soluţie SQL-92/DB2/Oracle şi la problema:

Extrageţi clienţii pentru care există cel puţin câte o factură emisă în fiecare zi.

```
SELECT DISTINCT DenCL FROM CLIENŢI Cl,
   FACTURI FI WHERE Cl.CodCl=Fl.CodCI
   AND EXISTS (SELECT 1 FROM FACTURI
   F2 WHERE F2.CodCI = FI.CodCI GROUP BY
   F2.CodCI
   HAVING COUNT(DISTINCT F2.DataFact) =
         (SELECT COUNT(DISTINCT DataFact)
         FROM FACTURI)
   )
```

Ultima consultare (de jos), cea din clauza HAVING, calculează numărul total de zile în care sunt vânzări, număr care este 5. Subconsultarea „mijlocie" determină dacă numărul de zile de vânzări pentru clientul curent (CI. CodCl) este egal cu 5 (rezultatul subconsultării de jos).

Care sunt cele mai mari cinci preţuri unitare la care s-au efectuat vânzări?

Această problemă a mai fost formulată în paragraful 5.4. Revenim cu o soluţie nou-nouţă, bazată pe subinterogări corelate.

```
SELECT PretUnit FROM
LINIIFACT LF1 WHERE 5
>
    (SELECT COUNT(DISTINCT LF2.PretUnit)
    FROM LINIIFACT LF2
    WHERE LF2.PretUnit > LF1.PretUnit)
ORDER BY PretUnit DESC
```

Ideea este grozav de ingenioasă (păcat că nu este a mea): se parcurge linie cu linie prima instanţă a tabelei LINIIFACT - LF1. Pentru fiecare linie de LF1 se numără în a doua instanţă a LINIIFACT - LF2 câte linii prezintă preţul unitar mai mare decât cel de pe linia curentă din LF1. Dacă numărul calculat prin COUNT (*) este mai mic decât 5, atunci în rezultat se extrage valoarea LF1. PretUnit.

Cum nici eu n-am înţeles prea bine explicaţiile de mai sus, să analizăm interogarea mai pe îndelete.

- *Pas 1:* se ia în discuţie primul tuplu din LINIIFACT (LF1). PretUnit are valoarea 10000. Se calculează numărul valorilor distincte ale PretUnit din LF2 pentru care LF2 . PretUnit > LF1. PretUnit. Acest număr este 8. Prin urmare, în LINIIFACT există opt preţuri unitare peste 10000. întrucât 8 > 5, primul tuplu din LF1 **nu** este inclus în rezultat.
- *Pas 2:* se „încarcă" al doilea tuplu din LINIIFACT (LF1). PretUnit are valoarea 10500. Funcţia COUNT din subconsultare întoarce 6 — în LINIIFACT există şase linii cu preţuri unitare peste 10600. Cum 6 > 5, nici al doilea tuplu din LF1 **nu** este inclus în rezultat.
- *Pas 3:* se „încarcă" al treilea tuplu din LINIIFACT (LF1), în care PretUnit este 6500. COUNT întoarce 15, deci nici al treilea tuplu din LF1 **nu** este inclus în rezultat.

- *Pas 8:* pe această linie PretUnit este 15800. COUNT întoarce 0; prin urmare, valoarea 15800 va face parte din rezultat.

în total sunt 22 de paşi, corespunzători tuplurilor din LINIIFACT. Păcat că VFP nu suportă asemenea găselniţă, sau cel puţin aşa spune mesajul care ne întâmpină la execuţia acestei interogări.

Subconsultări dublu corelate

Dubla corelare constituie o facilitate bine ocolită de către dezvoltatorii de aplicaţii (eu, unul, am reuşit s-o evit cu brio până în acest paragraf). Sheryl Larsen estima în 1998 că mai puţin de 10% dintre dezvoltatorii de aplicaţii în DB2 aveau cunoştinţă de interogările dublu corelate. Mărturisesc că în articolul lui Sheryl[56] am găsit cea mai bună explicaţie a acestui mecanism.

în cazul corelării simple, scenariul de execuţie este unul de tip *top-bottom-top*\ la corelarea dublă lucrurile stau cu mult mai interesant: *top-bottom-middle-bottom-middle-top.* Curat ca lacrima şi explicit ca reforma la români!

Dacă n-ar fi fost în joc diviziunea relaţională, n-am fi ajuns aici cu „hermeneutica" SQL. Revenim la cele două tabele, Rl şi R2, pe care le-am folosit în capitolul 2 la prezentarea diviziunii relaţionale. Valorile lui X care se găsesc în Rl în combinaţie cu toate valorile lui Y din R2 pot fi aflate şi prin interogarea următoare:

```
    SELECT DISTINCT X FROM Rl Tl_l WHERE NOT EXISTS (SELECT 1 FROM R2
WHERE NOT EXISTS
    (SELECT 1 FROM
    Rl Tl_2
    WHERE Tl_2.X=T1_1. X AND T1_2.Y=T2.Y )
)
```

Execuţia frazei SQL se derulează astfel:

Faza 1:

- *Pas 1.1: se* „încarcă" primul tuplu din Tl_l: (xl, yl) şi primul tuplu dinR2: yl.
  - *Pas 1.2.1:* se testează dacă există în Tl_2 o linie în care X=xl (Tl_l .X) şi Y=yl (R2 . Y). Există!, astfel încât se trece la următoarea linie din R2.
  - *Pas 1.2.2:* se testează dacă există în Tl_2 o linie în care X=xl (Tl_l .X) şi Y=y2 (R2 . Y). Există!, deci se „avansează" încă o linie în R2.
  - *Pas 1.2.3:* se testează dacă există în Tl_2 o linie în care X=xl (Tl_l .X) şi Y=y3 (R2 . Y). Există!, deci se „avansează" încă o linie în R2.
  - *Pas 1.2.4:* se testează dacă există în Tl_2 o linie în care X=xl (Tl_l .X) şi Y=y5 (R2 . Y). Există!, deci se „avansează" încă o linie în R2.
  - *Pas 1.2.5:* se testează dacă există în Tl_2 o linie în care X=xl (Tl_l .X) şi Y=y5 (R2 . Y). Există!, deci se încearcă încărcarea următoarei linii din R2.
  - *Pas 1.2.6:* întrucât am ajuns la sfârşitul tabelei R2 şi toate valorile din R2 s-au regăsit în Tl_2 în combinaţie cu xl,
- *Pas 1.3.:* subconsultarea de jos (*bottom*) întoarce TRUE către subconsultarea din mijloc (*middle*). Subconsultarea din mijloc recepţionează rezultatul SELECT-ului de jos şi îl pasează cu aceeaşi valoare logică sau cu valoare schimbată (dacă apare NOT) interogării principale. în această situaţie, SELECT-ul mijlociu primeşte de la cel de jos valoarea logică TRUE, însă, deoarece operatorul de conexiune mijloc-jos este NOT EXISTS, va întoarce SELECT-ului principal FALSE.
- *Pas 1.4.:* fraza SELECT principală recepţionează valoarea logică FALSE de la subconsultarea mijlocie. Dar operatorul de conexiune sus-mijloc este NOT EXISTS, iar FALSE-ul este transformat în TRUE, iar **valoarea xl va fi inclusă în rezultat!**

Faza 2:

- *Pas 2.1:* se „încarcă" al doilea tuplu din TI_____ 1: (x2, yl) şi primul tuplu din R2: yl.
  - *Pas 2.2.1:* se testează dacă există în Tl_2 o linie în care X=x2 (Tl_l .X) şi Y=yl (R2 . Y). Există!, astfel încât se trece la următoarea linie din R2.
  - *Pas 2.2.2:* se testează dacă există în Tl_2 o linie în care X=x2 (Tl_l .X) şi Y=y2 (R2 . Y). Nu există!, iar încărcarea celorlalte linii din R2 este abandonată şi
- *Pas 2.3:* subconsultarea de jos (*bottom*) întoarce FALSE către subconsultarea din mijloc (i*middle*). SELECT-ul mijlociu recepţionează valoarea logică FALSE şi, datorită operatorului NOT EXISTS, „pasează" SELECT-ului principal TRUE.
- *Pas 2.4:* fraza SELECT principală recepţionează valoarea logică TRUE de la subconsultarea mijlocie, pe care, la rândul său, o transformă în FALSE, iar **valoarea** x2 **nu va fi inclusă în rezultat!**

Lucrurile se continuă cu încă 14 faze, una pentru fiecare linie din Rl. Clauza DISTINCT asigură preluarea în rezultat a fiecărei valori o singură dată.

Dacă privim mai îndeaproape fraza SELECT, observăm că, de fapt, aceasta răspunde la întrebarea: pentru care dintre valorile lui X nu există nici un Y care să nu apară cu X-ul respectiv în combinaţie, pe (măcar) o linie în Rl.

In ce zile s-au vândut şi produsul cu denumirea „ Produs 1 ", şi cel cu denumirea „ Produs 2 "?
Soluţia SQL-99/DB2/Oracle bazată pe corelare dublă este probabil cea mai complicată, mult prea complicată pentru aşa o bagatelă de problemă:

```
SELECT DISTINCT DataFact
FROM FACTURI FI, LINIIFACT LF1, PRODUSE PI WHERE Fl.NrFact = LF1.NrFact
AND LF1.CodPr=Pl.CodPr AND NOT EXISTS (SELECT 1 FROM PRODUSE PI
    WHERE DenPr IN ('Produs 1',          'Produs 2') AND
          NOT EXISTS
            (SELECT 1
          FROM FACTURI F2, LINIIFACT LF2, PRODUSE P2 WHERE
    F2.NrFact = LF2.NrFact AND LF2.CodPr=P2.CodPr AND .
    F2.DataFact=Fl.DataFact AND P2.CodPr=Pl.CodPr
        )
    )
```

Extrageţi clienţii pentru care există cel puţin câte o factură emisă în fiecare zi.

Reluăm această problemă de la diviziunea relaţională ce poate fi rezolvată şi prin SELECT-ul următor:

```
SELECT DISTINCT DenCl FROM
CLIENŢI CI, FACTURI FI WHERE
CI.CodCl=Fl.CodCl AND NOT EXISTS
(SELECT 1
```

```
                    FROM FACTURI F2 WHERE NOT EXISTS (SELECT 1
                  FROM CLIENŢI C3, FACTURI F3 WHERE C3.CodCl=F3.CodCI AND
                                C3.CodCl=Cl.CodCI AND
                   F3.DataFact = F2.DataFact
        )
    )
```

Toate soluţiile ce întrebuinţează dubla corelare de până acum au o concurenţă care le pune în umbră. Variantele ne-bazate pe dubla corelare sunt mai simple, atât ca dimensiune, cât şi ca mod de înţelegere. Este timpul să scoatem din mânecă o problemă de diviziune relaţională care să demonstreze adevărata forţă a dublei corelări şi să justifice generosul efort intelectual pentru a o înţelege:

Ce facturi conţin măcar produsele din factura 1112?

```
SELECT DISTINCT NrFact FROM
LINIIFACT LF1 WHERE NOT
EXISTS (SELECT 1
          FROM LINIIFACT LF2 WHERE NrFact = 1112 AND NOT EXISTS (SELECT 1
                                  FROM LINIIFACT LF3 ,
      WHERE LF3.CodPr = LF2.CodPr AND
        LF3.NrFact=LFl.NrFact )
    )
```

LINIIFACT are 22 de linii, deci execuţia se desfăşoară în 22 de faze, dintre care prezentăm două:

Faza 1:

- *Pas 1.1:* se „încarcă" primul tuplu din LF1: (1111, 1, 1, 50, 10000) şi primul tuplu din LF2 care îndeplineşte condiţia NrFact = 1112: (1112, 1, 2, 80, 10300).
- *Pas 1.2.1:* se testează dacă există în LF3 măcar o linie în care NrFact=llll (LF1. Nrfact) şi CodPr=2 (LF2. CodPr). Există (a doua), astfel încât se trece la următoarea linie din LF2 care îndeplineşte condiţia NrFact = 1112: (1112, 1, 3, 40, 7500).
- *Pas 1.2.2:* se testează dacă există în LF3 măcar o linie în care NrFact=llll (LF1. Nrfact) şi CodPr=3 (LF2 . CodPr). Nu există!, iar
- *Pas 1.3:* subconsultarea de jos (*bottom)* întoarce FALSE către subconsultarea din mijloc (*middle*). SELECT-ul mijlociu recepţionează valoarea logică FALSE şi, datorită operatorului NOT EXISTS, „pasează" SELECT-ului principal TRUE.

® *Pas 1.4:* fraza SELECT principală recepţionează valoarea logică TRUE de la subconsultarea mijlocie, pe care, la rândul său, o transformă în FALSE, iar **valoarea 1111 nu va fi inclusă în** rezultat!

Faza 16:

- *Pas 16.1:* se „încarcă" tuplul 16 din LF1: (1! 19, 1, 2, 35, 10900) şi primul tupiu din LF2 care îndeplineşte condiţia Nr Fact = 1112: (1112, 1, 2, 80, 10300).

® *Pas 16.2.1:* se testează dacă există în LF3 măcar o linie în care NrFact=1119 (LF1. Nrfact) şi CodPr=2 (LF2. CodPr). Există (tuplul 16), astfel încât se trece la următoarea linie din LF2 care îndeplineşte condiţia NrFact = 1112: (1112, I, 3, 40, 7500).

® *Pas 16.2.2:* se testează dacă există în LF3 măcar o linie în care NrFact=1119 (LF1. Nrfact) şi CodPr=3 (LF2. CodPr). Există (tuplul 17), astfel încât se încearcă trecerea la următoarea linie din LF2 care îndeplineşte condiţia NrFact = 1112. Cum nu mai există nici o asemenea linie,

[9] *Pas 16.3:* subconsultarea de jos (*bottom)* întoarce TRUE către subconsultarea din mijloc *(middle),* care, la rândul său, întoarce FALSE SELECT-uiui principal.

® *Pas 16.4:* fraza SELECT principală recepţionează valoarea logică FALSE de la subconsultarea mijlocie, prin negarea căreia obţine TRUE, iar valoarea 1119 va fi inclusă în rezultat!

Fraza SELECT analizată răspunde, de fapt, la întrebarea: Pentru care facturi nu există nici un produs ce apare în factura 1112, dar nu este prezent în factura respectivă?

Fără dubla corelare, pot fi formulate şi soluţii bazate pe MINUS (EXCEPT) şi IN, precum:

```
SELECT      NrFact
FROM    LINIIFACT
MINUS
   SELECT DISTINCT LF1.NrFact FROM
   LINIIFACT LF1, LINIIFACT LF2 WHERE
   LF2.NrFact = 1112 AND
      (LF1.Nrfact, LF2.CodPr) NOT IN (SELECT
         NrFact, CodPr FROM LINIIFACT)
```

(soluţie Oracle; în DB2 se înlocuieşte MINUS cu EXCEPT) sau cu aceeaşi logică, dar ce utilizează tandemul MINUS (EXCEPT) - EXISTS:

```
SELECT      NrFact
FROM    LINIIFACT
MINUS
   SELECT DISTINCT LFl.NrFact FROM LINIIFACT
   LF1, LINIIFACT LF2 WHERE LF2.NrFact = 1112
   AND NOT EXISTS (SELECT 1
      FROM LINIIFACT LF3 WHERE
      LF3.NrFact=LFl.NrFact AND
      LF3.CodPr=LF2.CodPr)
```

Deşi ultimele două soluţii sunt comparabile, ca dimensiune, cu varianta bazată pe dubla corelare, faptul că logica lor presumme produsul cartezian a două instanţe ale tabelei LINIIFACT. tabelă
care, în timp, acumulează un uriaş număr de înregistrări, le pune în umbră.
. intru a respecta adevărul istoric, trebuie spus câ se poate recurge la artificii non-diviziune relaţională, precum cel următor:

```
SELECT DISTINCT
      NrFact FROM
LINIIFACT WHERE
 CodPr IN (SELECT
      CodPr FROM
LINIIFACT WHERE
     NrFact = 1112)
                  GROUP BY NrFact HAVING COUNT(*) =
(SELECT COUNT(CodPr)
FROM LINIIFACT WHERE
NrFact = 1112)
```

Care sunt clienţii cărora li s-au vândut cel puţin produsele vândute clientului CLIENT 4?

```
SELECT DenCl FROM CLIENŢI Cl WHERE NOT EXISTS (SELECT 1
FROM CLIENŢI C2, FACTURI F2, LINIIFACT LF2 WHERE
F2.NrFact=LF2.NrFact AND C2.CodCl=F2.CodCI AND DenCl='Client 4'
AND NOT EXISTS (SELECT 1
      FROM FACTURI F3, LINIIFACT LF3 WHERE
   F3.NrFact=LF3.NrFact AND Cl.CodCl=F3.CodCI AND
   LF3.CodPr=LF2.CodPr))
```

Analizând enunţul problemei „*Care sunt clienţii...* rezultă că atributul pentru corelarea consultării principale *{top)* şi celei de jos (*bottom*) este CodCI, iar din partea condiţională a enunţului *cel puţin produsele...* ” pentru corelarea mijloc (*middle*) - jos (*bottom*) se utilizează atributul CodPr.

6.2. Subconsultări în clauza FROM

Posibilitatea de a defini tabele ad-hoc, în clauza FROM, **este un privilegiu pe care şi-l pot permite numai SGBD-urile profesionale, serverele de baze de date. Ne despărţim, aşadar, din nou de VFP pentru o vreme.**

Care sunt valorile facturate şi încasate, precum şi situaţia („fără nici o încasare ”, „ încasată parţial" sau „ încasată total”) pentru fiecare factură?

- Soluţia SQL-92/DB2:

```
SELECT VINZARI.Nrfact, Facturat, VALUE(încasat,0) AS încasat, Facturat - VALUE(încasat,0)
        AS Diferenţa,
    CASE
        WHEN VALUE(încasat, 0) = 0
            THEN 'Fara nici o incasare'
        WHEN Facturat > VALUE(încasat,0)
            THEN 'încasata parţial'
        ELSE 'ÎNCASATA TOTAL'
    END AS Situatiune
```

```
FROM
    (SELECT NrFact,
          SUM(Cantitate * PretUnit * (1+ProcTVA))
              AS Facturat FROM LINIIFACT LF INNER JOIN
     PRODUSE P ON LF.CodPr = P.CodPr GROUP BY NrFact)
     VINZARI
    LEFT OUTER JOIN
          (SELECT NrFact, SUM(Transa) AS încasat
          FROM INCASFACT
          GROUP BY NrFact) INCASARI
    ON VINZARI.NrFact = INCASARI.NrFact
```

- Soluţia Oracle 8:

```
SELECT VINZARI.Nrfact, Facturat, NVL(încasat,0) as încasat, Facturat - NVL(încasat,0) as
    Diferenţa,
    DECODE (SIGN(Facturat-NVL(încasat,0)) ,
        0, 'ÎNCASATA TOTAL',
        -1,'încasat mai mult decit facturat ! ! ! ' ,
        DECODE(NVL(încasat,0),
            0, 'Fara nici o incasare',
            'încasata parţial')
        )
    AS Situatiune FROM
    (SELECT NrFact,
        SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat FROM LINIIFACT LF,
     PRODUSE P WHERE LF.CodPr = P.CodPr GROUP BY NrFact) VINZARI,
    (SELECT NrFact, SUM(Transa) AS încasat
    FROM INCASFACT
    GROUP BY NrFact) INCASARI
WHERE VINZARI.NrFact = INCASARI.NrFact (+)
```

în clauza FROM a frazei SELECT principale au fost definite două tabele în toată regula, VÂNZĂRIşi ÎNCASĂRI - vezi figura 6.2. Prima conţine valoarea totală a fiecărei facturi, în timp ce a doua conţine valoarea încasată. Aceste două tabele sunt joncţionate extern după atributul Nr Fact. În Oracle 8, în locul structurii CASE s-a folosit funcţia DECODE.

VINZARI

NRFACT	FACTURAT
1111	5399625.00
1112	1337560.00
1113	1160250.00
1114	6786570.00
1115	1651125.00
1116	1383375.00
1117	2320500.00
1118	2052750.00
1119	7242935.00
1120	1066240.00
1121	5438300.00

INCASARI

NRFACT	INCASAT
1111	5399625
1112	487705
1113	1160250
1117	2320500
1118	2052750
1120	731557

Figura 6.2. Rezultatele celor două subconsultări din clauza FROM

Cele două variante nu sunt echivalente. Soluţia Oracle 8 plus, şi afişează, în situaţia, oarecum anormală, când valoarea încasată a unei unei facturi o depăşeşte pe cea facturată.

Să se obţină sporurile de noapte pentru al doilea trimestru al anului 2000, atât lunar, cât şi cumulai

- Soluţia SQL-92/DB2:

```
SELECT SL1.Marca, NumePren,
    VALUE(SL1.SporNoapte,0) AS Spor_Noapte_Aprilie,
    VALUE(SL2.SporNoapte,0) AS Spor_Noapte_Mai,
    VALUE(SL3.SporNoapte,0) AS Spor_Noapte_Iunie,
    VALUE(SL1.SporNoapte,0) + VALUE(SL2.SporNoapte,0)+
       VALUE(SL3.SporNoapte,0) AS Spor_Noapte_Trim_II
FROM
    (SELECT PERSONAL2.Marca, NumePren, SporNoapte FROM
    PERSONAL2
          LEFT OUTER JOIN SPORURI
             ON PERSONAL2.Marca=SPORURI.Marca AND
    An=2000 AND Luna =4) SL1 INNER JOIN (SELECT
    PERSONAL2.Marca, SporNoapte FROM PERSONAL2
    LEFT OUTER JOIN SPORURI ON
    PERSONAL2.Marca=SPORURI.Marca AND An=2000
    AND Luna =5) SL2 ON SL1.Marca=SL2.Marca INNER
    JOIN
    (SELECT PERSONAL2.Marca, SporNoapte FROM PERSONAL2
LEFT OUTER JOIN SPORURI ON PERSONAL2.Marca=SPORURI.Marca
AND An=2000 AND Luna =6) SL3 ON SL1.Marca=SL3.Marca ORDER
BY NumePren
```

® Soluţia Oracle 8:

```
SELECT SLl.Marca, NumePren,
    NVL(SLl.SporNoapte,0) AS Spor_Noapte_Aprilie,
    NVL(SL2.SporNoapte,0) AS Spor_Noapte_Mai,
    NVL(SL3.SporNoapte,0) AS Spor_Noapte_Iunie,
    NVL(SLl.SporNoapte,0) + NVL(SL2.SporNoapte,0)+
    NVL (SL3 . SporNoapte, 0 ) AS Spor_Noapte__Trim_II
FROM
    (SELECT PERSONAL2.Marca, NumePren, SporNoapte
    FROM PERSONAL2, SPORURI
    WHERE PERSONAL2.Marca=SPORURI.Marca (+)
       AND An    (+) =   2000 AND Luna   (+)=4) SLl,
    (SELECT PERSONAL2.Marca, SporNoapte
    FROM PERSONAL2, SPORURI
    WHERE PERSONAL2.Marca=SPORURI.Marca (+)
       AND An    (+) =   2000 AND Luna   (+)=5) SL2,
    (SELECT PERSONAL2.Marca, SporNoapte
    FROM PERSONAL2, SPORURI
    WHERE PERSONAL2.Marca=SPORURI.Marca (+)
       AND An    (+) =   2000 AND Luna   (+) =6) SL3
WHERE SLl.Marca=SL2.Marca AND SLl.Marca=SL3.Marca
ORDER BY NumePren
```

Clauza FROM principală „calculează" trei tabele ce conţin sporurile de noapte ale lunilor aprilie (SLl), mai (SL2) şi iunie (SL3) 2000 ale fiecărui angajat (indiferent de data angajării acestuia) - figura 6.3. Cele trei tabele sunt joncţionate după atributul Marca.

Revenim la diviziunea relaţională. Succesiunea de paşi prezentată în figura 2.28 se poate realiza şi prin soluţia SQL-92/DB2:

```
SELECT DISTINCT
X FROM Rl
WHERE X NOT IN
   (SELECT DISTINCT PROD_CART.X                    '
   FROM (SELECT DISTINCT Rl.X, R2.Y FROM R1,R2) PROD_CART
   LEFT OUTER JOIN Rl ON PROD_CART.X=R1.X AND
   PROD_CART.Y=R1.Y WHERE Rl.X IS NULL)
```

SL1

MARCA	NUMEPREN	SPORNOAPTE
1	ANGAJAT 1	0
2	ANGAJAT 2	450000
3	ANGAJAT 3	560000
4	ANGAJAT 4	
5	ANGAJAT 5	
6	ANGAJAT 6	
7	ANGAJAT 7	
8	ANGAJAT 8	
9	ANGAJAT 9	
10	ANGAJAT 10	

SL2

MARCA:	SPORNOAPTE
1	C
2	45000C
3	C
4	
5	
6	
7	
8	
9	
10	C

SL3

MARCA	SPORNOAPTE
1	0
2	0
3	
4	150000
5	150000
6	
7	
8	
9	
10	80000

Figura 6.3. Rezultatele celor trei subconsultări

şi SQL-92/DB2 plus Oracle:

```
SELECT DISTINCT X
FROM Rl
WHERE X NOT IN
   (SELECT DISTINCT PROD_CART.X
      FROM (SELECT DISTINCT Rl.X, R2.Y FROM R1,R2) PROD_CART,
         Rl
   WHERE PROD_CART.X=R1.X (+) AND PROD_CART.Y=R1.Y (+)
      AND Rl.X IS NULL)
```


Putem încerca însă şi ceva mai elegant. Ce ziceţi de soluţia:

```
SELECT DISTINCT X
FROM                                                    '
   (SELECT X, COUNT(Y) AS Nr FROM Rl GROUP BY X) TEMP1,
   (SELECT COUNT(Y) AS Nr FROM R2) TEMP2 WHERE
TEMP1.Nr=TEMP2.Nr
```

Prima tabelă, TEMP1, conţine numărul valorilor lui `Y` pentru fiecare `X` din Rl, iar `a` doua, TEMP2, numai numărul total al valorilor lui `Y` din R2 - figura 6.4.

TEMP1

	NR
X1	5
x2	2
x3	5
x4	2
xS	2

TEMP2

NR
5

Figura 6.4. Tabelele intermediare (ad-hoc) TEMP1 şi TEMP2

Care sunt clienţii pentru care există cel puţin câte o factură emisă în fiecare zi?

Este exemplul 22 din algebra relaţională (figura 2.27), formulat pentru ilustrarea operatorului: diviziune.

- Soluţia 1 - DB2 (şi Oracle, operând modificările în sintaxa joncţiunii externe):

```
SELECT DISTINCT DenCl FROM CLIENŢI WHERE CodCI NOT IN
   (SELECT DISTINCT PROD_CART.CodCI FROM
          (SELECT DISTINCT CLIENŢI.CodCI, FACTURI.DataFact FROM
          CLIENŢI, FACTURI) PROD_CART LEFT OUTER JOIN FACTURI
          ON PROD_CART.CodCl=FACTURI.CodCI AND
   PROD_CART.DataFact=FACTURI.DataFact WHERE FACTURI.CodCI IS
   NULL)
```

- Soluţia 2 (cea preferată) - DB2 şi Oracle:

```
SELECT DISTINCT DenCl
FROM
    (SELECT CodCl, COUNT(DISTINCT DataFact) AS Nr
    FROM FACTURI
    GROUP BY CodCl) TEMP1,
    (SELECT COUNT(DISTINCT DataFact) AS Nr FROM
    FACTURI) TEMP2,
    CLIENŢI
WHERE TEMP1.Nr=TEMP2.Nr AND TEMP1.CodCl=CLIENTI.CodCl
```

In ce zile s-au vândut şi produsul cu denumirea,, Produs I ", şi cel cu denumirea,, Produs 2 ''? Este (tot) exemplul 23 din capitolul 2.

- Soluţia 1-DB2 (şi Oracle, înlocuind `LEFT OUTER JOIN`):

```
SELECT DISTINCT DataFact
FROM FACTURI F, LINIIFACT LF, PRODUSE P WHERE
F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr AND DenPr IN
('Produs 1', 'Produs 2')
   AND DataFact NOT IN
       (SELECT DISTINCT PROD_CART.DataFact
       FROM
         (SELECT DISTINCT DataFact, CodPr FROM
         FACTURI, PRODUSE
         WHERE DenPr IN ('Produs 1', 'Produs 2')
       ) PROD_CART
   LEFT OUTER JOIN
       (SELECT F.DataFact, LF.CodPr FROM FACTURI F,
       LINIIFACT LF, PRODUSE P WHERE
       F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr ) TEMP1

       ON PROD_CART.DataFact=TEMPl.DataFact AND
   PROD_CART.CodPr=TEMP1.CodPr WHERE TEMP1.DataFact
   IS NULL)
```

- Soluţia 2 - DB2/Oracle:

```
SELECT DISTINCT DataFact
FROM
    (SELECT F.DataFact, COUNT(DISTINCT LF.CodPr) AS Nr
    FROM FACTURI F, LINIIFACT LF, PRODUSE P WHERE
    F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr AND DenPr IN
    ('Produs 1',. 'Produs 2')
    GROUP BY F.DataFact) TEMP1 WHERE Nr = 2
```

Din nou, soluţia a doua contrastează evident, prin simplitate, cu prima. Tabela TEMP1 conţine, pentru fiecare dată calendaristică, numărul de produse, dintre *Produs 1* şi *Produs 2* (1 sau 2) care au fost vândute în ziua respectivă.

Ce facturi conţin măcar produsele din factura 1112?

```
SELECT DISTINCT NrFact FROM
   (SELECT NrFact, COUNT(*) AS NrProd FROM
   LINIIFACT WHERE CodPr IN
        (SELECT     CodPr
        FROM    LINIIFACT
        WHERE   NrFact   =
        1112)
        GROUP BY NrFact
   ) Tl,
   (SELECT COUNT(CodPr) AS NrP1112 FROM
LINIIFACT WHERE NrFact = 1112) T2 WHERE
Tl.NrProd = T2.NrP1112
```

T2 conţine numărul produselor din factura 1112. Tl conţine, pentru fiecare factură, câte produs; sunt comune acesteia şi facturii 1112. Prin joncţiunea Tl cu T2 prin condiţia `Tl. NrProd = T2 . NrP1112,` se extrag acele linii din Tl care au acelaşi număr de produse prezente în factin 1112 ca şi aceasta.

Care sunt clienţii cărora li s-au vândut cel puţin produsele vândute clientului CLIENT 4?

```
SELECT DISTINCT DenCl
```

```
FROM (
   SELECT DenCl, COUNT(*) AS NrProd FROM CLIENŢI C,
FACTURI F, LINIIFACT LF WHERE C.CodCl=F.CodCI AND
F.NrFact=LF.Nrfact AND CodPr IN (SELECT CodPr
      FROM CLIENŢI C, FACTURI F, LINIIFACT LF WHERE
      C.CodCl=F.CodCI AND F.NrFact=LF.Nrfact AND
      DenCl='Client 4' )
      GROUP BY DenCl )
   Tl,
   (
   SELECT COUNT(CodPr) AS NrProd FROM CLIENŢI C, FACTURI
F, LINIIFACT LF WHERE C.CodCl=F.CodCI AND
F.NrFact=LF.Nrfact AND DenCl='Client 4'
   ) T2
WHERE Tl.NrProd = T2.NrProd
```

Logica soluţiei este cât se poate se asemănătoare precedentei, doar că T2 conţine numl-J produselor vândute' clientului 4, iar în Tl, pe fiecare linie se găseşte un client şi numSnd produselor vândute clientului 4 care i-au fost vândute şi lui.

Să se afişeze câte facturi sunt: neîncasate deloc, încasate parţial şi încasate total?

Prezentăm numai soluţia DB2.

```
SELECT
   CASE
      WHEN VALUE(încasat, 0) = 0
         THEN 'Fara nici o incasare'
      WHEN Facturat > VALUE(încasat,0)
         THEN 'încasata parţial'
      ELSE 'ÎNCASATA TOTAL'
   END AS Situatiune, COUNT (*) AS Nr
FROM
   (
      SELECT NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS
      Facturat FROM LINIIFACT LF
         INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr
   GROUP BY NrFact ) VINZARI
   LEFT OUTER JOIN
   (
   SELECT NrFact, SUM(Transa) AS încasat
   FROM INCASFACT GROUP BY NrFact ) INCASARI
         ON VINZARI.NrFact = INCASARI.NrFact
GROUP BY CASE
         WHEN VALUE(încasat,0) = 0
            THEN 'Fara nici o incasare'
         WHEN Facturat > VALUE(încasat,0)
            THEN 'încasata parţial'
         ELSE 'ÎNCASATA TOTAL'
   END
```

Joncţiunea externă la stânga dintre VÂNZĂRI şi ÎNCASĂRI este completată `de` o structură alternativă multiplă – `CASE` sau `DECODE` (în Oracle).

Care angajaţi au salariul tarifar egal cu cel al ANGAJAT2?

```
SELECT NumePren FROM
   (SELECT NumePren, SalTarifar FROM
   PERSONAL2) TEMP1,
   (SELECT Saltarifar FROM PERSONAL2
   WHERE NumePren='ANGAJAT 2') TEMP2 WHERE
TEMP1.SalTarifar = TEMP2.SalTarifar
```

Care este ziua în care s-au emis cele mai multe facturi?
Din păcate (sau din fericire!), interogarea:

```
SELECT TEMP1.DataFact, TEMPl.Nr FROM
    (SELECT DataFact, COUNT(Nrfact) AS Nr FROM
    FACTURI
    GROUP BY DataFact) TEMP1,
    (SELECT DataFact, COUNT(Nrfact) AS Nr FROM
    FACTURI
    GROUP BY DataFact) TEMP2 WHERE TEMPl.Nr >= ALL
(SELECT Nr FROM'TEMP2)
```


nu este operaţională. Aceasta înseamnă că o tabelă definită ad-hoc într-o frază SELECT nu este recunoscută în subconsultări. Se poate, totuşi, utiliza varianta:

```
SELECT DataFact, COUNT(*) AS Nr_Facturi FROM FACTURI GROUP BY DataFact
HAVING COUNT(*) =
    (SELECT MAX(Nr)
    FROM
        (SELECT DataFact, COUNT(NrFact) AS Nr FROM
        FACTURI GROUP BY DataFact) TEMP1)
```

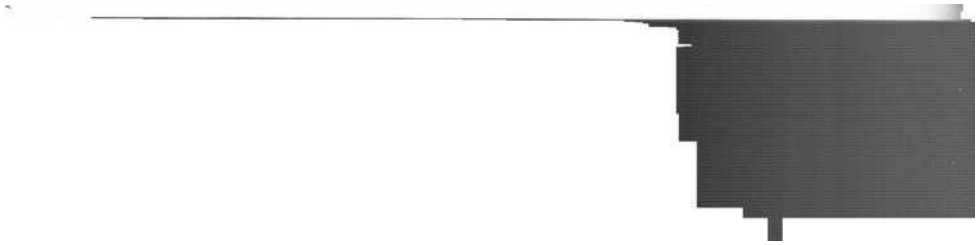
în subconsultare s-a definit tabela intermediară TEMP1, al cărei atribut Nr este folosit în funcţia MAX din clauza SELECT. O altă variantă bazată pe subconsultare în FROM este şi:

```
SELECT DataFact, NrFacturi FROM
    (SELECT DataFact, COUNT(*) AS NrFacturi FROM FACTURI
    GROUP BY DataFact) FACT_ZILE,
    (SELECT MAX(COUNT(*)) AS NrMax FROM FACTURI
GROUP BY DataFact) MAX WHERE NrFacturi=NrMax
```

Care este clientul care a cumpărat cele mai multe produse?
După modelul interogării anterioare, formulăm două soluţii generale (SQL/DB2/Oracle) bazate pe subconsultări în clauza FROM.

- Soluţia 1:

```
SELECT DenCl, COUNT(DISTINCT CodPr) AS Nr_Produse
FROM CLIENŢI, FACTURI, LINIIFACT
WHERE CLIENŢI.CodCl=FACTURI.CodCI AND
FACTURI.NrFact=LINIIFACT.NrFact
GROUP BY DenCl
```



```
HAVING COUNT(DISTINCT CodPr) =
   (SELECT MAX(Nr)
   FROM
       (SELECT CodCl, COUNT(DISTINCT CodPr) AS Nr
       FROM FACTURI, LINIIFACT WHERE
       FACTURI.NrFact=LINIIFACT.NrFact GROUP BY CodCl
       ) TEMP1
   )
```

® Soluţia 2:

```
SELECT DenCl, CL_PROD.NrProd FROM
   (SELECT DenCl, COUNT(DISTINCT CodPr) AS NrProd FROM
   CLIENŢI, FACTURI, LINIIFACT WHERE
   CLIENŢI.CodCl=FACTURI.CodCl AND
   FACTURI.NrFact=LINIIFACT.NrFact GROUP BY DenCl )
```

```
   CL_PROD,
   (SELECT MAX(COUNT(DISTINCT CodPr)) AS NrProd FROM
FACTURI, LINIIFACT WHERE
FACTURI.NrFact=LINIIFACT.NrFact GROUP BY CodCl) CL_MAX
WHERE CL_PROD.NrProd=CL_MAX.NrProd
```

Care este judeţul în care berea s-a vândut cel mai bine?

```
SELECT Judeţ, SUM(Cantitate * PretUnit * (1+ProcTVA))
   AS Vinzari_Bere FROM JUDEŢE J, LOCALITATI L, CLIENŢI
C, FACTURI F,
   LINIIFACT LF, PRODUSE P WHERE J.Jud=L.Jud AND
L.CodPost=C.CodPost AND C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND
LF.CodPr=P.CodPr AND Grupa='Bere'
GROUP BY Judeţ
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) =
   (SELECT MAX(Vinzari)
   FROM
           (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
           AS Vinzari FROM LOCALITATI L, CLIENŢI C, FACTURI
         F,
           LINIIFACT LF, PRODUSE P WHERE
         L.CodPost=C.CodPost AND C.CodCl=F.CodCl AND
         F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr AND
         Grupa='Bere'
         GROUP BY Jud
         ) TEMP1
   )
```

Care sunt clienţii cu valoarea vânzărilor peste medie?

- Soluţia 1:

```
SELECT DenCl, SUM(Cantitate * PretUnit * (1+ProcTVA))
      AS Vinzari
FROM CLIENŢI C, FACTURI F, LINIIFACT LF, PRODUSE P
WHERE C.CodCl=F.CodCI AND F.NrFact=LF.NrFact AND
LF.CodPr=P.CodPr GROUP BY DenCl
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) >=
   (SELECT Vinzari / NrClienti FROM
      (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
         AS Vinzari FROM LINIIFACT LF, PRODUSE P WHERE
      LF.CodPr=P.CodPr ) TEMP1,
      (SELECT COUNT(DISTINCT CodCI) AS NrClienti FROM
      FACTURI)
   TEMP2
   )
```

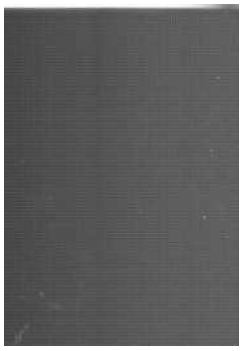
® Soluţia 2:

```
SELECT DenCl, VINZ_CL.Vinzari
FROM
   (SELECT DenCl, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS
        Vinzari
   FROM CLIENŢI C, FACTURI F, LINIIFACT LF, PRODUSE P WHERE
   C.CodCl=F.CodCI AND F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
   GROUP BY DenCl ) VINZ_CL,
   (SELECT DISTINCT Vinzari / NrClienti AS Medie_Vinz FROM
         (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
           AS Vinzari FROM LINIIFACT
         LF, PRODUSE P WHERE
         LF.CodPr=P.CodPr),
         (SELECT COUNT(DISTINCT CodCI) AS NrClienti FROM
         FACTURI)
         ) MEDIE_VINZ WHERE VINZ_CL.Vinzari >=
MEDIE_VINZ.Medie_Vinz
```

Care este factura cu cea mai mică valoare peste cea medie?

```
SELECT NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA))
```

```
     AS ValFact FROM LINIIFACT LF, PRODUSE P
WHERE LF.CodPr=P.CodPr
GROUP BY NrFact
```

```
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) =
   (SELECT MIN(ValFact)
   FROM
         (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
            AS ValFact FROM LINIIFACT LF,
   PRODUSE P WHERE LF.CodPr=P.CodPr GROUP BY
   NrFact) TEMP1 WHERE ValFact >
         (SELECT Vinzari / NrFacturi AS ValMedie FROM
            (SELECT SUM(Cantitate * PretUnit *
              (1 + ProcTVA)) AS Vinzari,
             COUNT(DISTINCT NrFact) AS NrFacturi
            FROM LINIIFACT LF, PRODUSE P WHERE
            LF.CodPr=P.CodPr ) TEMP1
         )
   )
```

Apelăm şi la o soluţie care să afişeze toate facturile cu valoarea peste medie, în ordinea crescătoare a acestei valori. Prima factură din rezultat - figura 6.5 - va fi răspunsul la întrebarea formulată.

```
SELECT NrFact, ValFact, ValMedie
FROM
   (SELECt NrFact, SUM(Cantitate * PretUnit * (1+ProcTVA))
         AS ValFact FROM LINIIFACT LF, PRODUSE P WHERE
   LF.CodPr=P.CodPr GROUP BY NrFact) TEMP1,
   (SELECT ROUND(Vinzari / NrFacturi,0) AS ValMedie FROM
         (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
         AS Vinzari,
         COUNT(DISTINCT NrFact) AS NrFacturi FROM LINIIFACT
LF, PRODUSE P WHERE LF.CodPr=P.CodPr) MEDIEI ) MEDII WHERE
ValFact > ValMedie ORDER BY ValFact
```

NRFACT	VALFACT	VALMEDIE
1111 5399625 3258112		
1121	5438300 3258112	
1114	6786570 3258112	
1119	7242935 3258112	

Care este judeţul cu vânzări imediat superioare judeţului Neamţ?

Soluţia precedentă este rezonabilă, dar nu răspunde punctual la întrebare. Drept care, la prezentul exemplu, ne punem problema afişării numai a judeţului şi valorii vânzărilor. Vom apela la un truc ieftin, pe care l-am mai utilizat. Iată interogarea DB2 şi rezultatul său în figura 6.6.

```
SELECT MIN (CAST (CAST (VINZ_JUD1.Vinzari
     AS DECIMAL (15,0)) AS CHAR (1.6)) CONCAT ' - '
     CONCAT VINZ_JUD1.Judeţ)
   AS Rezultat FROM
   (SELECT Judeţ, SUM(Cantitate * PretUnit * (1+ProcTVA))
        AS Vinzari
```

```
   FROM JUDEŢE J, LOCALITATI L, CLIENŢI C, FACTURI F, LINIIFACT LF,
PRODUSE P WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
C.CodCl=F.CodCI AND F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr GROUP
BY Judeţ) VINZ_JUD1,
   (SELECT Judeţ, SUM(Cantitate * PretUnit * (1+ProcTVA))
        AS Vinzari
   FROM JUDEŢE J, LOCALITATI L, CLIENŢI C, FACTURI F, LINIIFACT LF,
PRODUSE P WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
C.CodCl=F.CodCI AND F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr GROUP
BY Judeţ) VINZ_JUD2
WHERE VINZ_JUD1.Vinzari > VINZ_JUD2.Vinzari AND
VINZ JUD2.Judet='Neamţ'
```

REZULTAT
000000007242935. - Vaslui

Figura 6.6. Judeţul cu vânzări imediat superioare judeţului Neamţ

Tabelele ad-hoc VINZJUD1 şi VINZ JUD2 sunt identice şi conţin valoarea vânzărilor pentru fiecare judeţ. Fraza SELECT principală le theta-joncţionează, după expresia VINZ_JUD1 .Vinzari > VINZ_JUD2. Vinzari, în condiţiile în care această comparaţie se face pentru linia în care VINZ_JUD2 . Judet= ' Neamţ'. Rezultă că vor fi extrase acele judeţe (şi vânzările corespunzătoare) care au valoarea vânzărilor peste cea a judeţului Neamţ. Trucul ieftin de care vorbeam este plasat în clauza SELECT a interogării principale. Ideea este de a concatena valoarea vânzărilor cu denumirea judeţului. Cum minimul este decis de primul argument, cârpeala funcţionează!

O soluţie fără improvizaţii este următoarea:

```
SELECT Judeţ, SUM(Cantitate * PretUnit * (1+ProcTVA))
   AS Vinzari
FROM JUDEŢE J, LOCALITATI L, CLIENŢI C, FACTURI F,
   LINIIFACT LF, PRODUSE P
WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
    C.CodCl=F.CodCl
    AND F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr GROUP BY Judeţ
HAVING SUM(Cantitate * PretUnit * (1+ProcTVA)) =
    (SELECT MIN ( VINZ JUDI.Vinzari)
    FROM
         (SELECT Judeţ, SUM(Cantitate * PretUnit *
         (1+ProcTVA)) AS Vinzari FROM JUDEŢE J,
         LOCALITATI L, CLIENŢI C,
           FACTURI F, LINIIFACT LF, PRODUSE P WHERE
         J.Jud=L.Jud AND L.CodPost=C.CodPost AND
         C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND
         LF.CodPr=P.CodPr GROUP BY Judeţ) VINZ_JUD1,
    (SELECT Judeţ, SUM(Cantitate * PretUnit * (1+ProcTVA))
         AS Vinzari
    FROM JUDEŢE J, LOCALITATI L, CLIENŢI C, FACTURI F, LINIIFACT
LF, PRODUSE P WHERE J.Jud=L.Jud AND L.CodPost=C.CodPost AND
C.CodCl=F.CodCl AND F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr GROUP
BY Judeţ) VINZ_JUD2
    WHERE VINZ_JUD1.Vinzari > VINZ_JUD2.Vinzari AND
```

```
        VINZ_JUD2.Judet=1 Neamţ')
```

Prima variantă furnizează un răspuns incomplet dacă sunt la egalitate două sau mai multe judeţe care îndeplinesc condiţia.

Care este clientul cel mai datornic (care are cel mai mare rest de plată)?

Fondul problemei este asemănător celei precedente. în interogare obţinem o tabelă ad-hoc cu diferenţa de încasat pe clienţi (TEMP1) şi o alta ce conţine cea mai mare diferenţă de încasat pentru un client (TEMP2). Cele două tabele sunt joncţionate după diferenţă şi, în final, pentru a afla denumirea clientului, adăugăm injoncţiune tabela CLIENŢI.

- Soluţia DB2:

```
SELECT DenCl, Vinzari, Incasari, Delncasat FROM
    (SELECT FACTURAT.CodCl, Vinzari,
           VALUE(Incasari, 0) AS Incasari,
           Vinzari - VALUE(Incasari, 0) AS Delncasat
    FROM
    (SELECT CodCl, SUM(Cantitate * PretUnit *
           (1+ProcTVA)) AS Vinzari FROM FACTURI F, LINIIFACT
    LF, PRODUSE P WHERE F.NrFact=LF.NrFact AND
    LF.CodPr=P.CodPr GROUP BY CodCl) FACTURAT LEFT OUTER JOIN
  '(SELECT CodCI, SUM(Transa) AS Incasari FROM FACTURI F,
  INCASFACT I WHERE I.NrFact=F.NrFact GROUP BY CodCI) ÎNCASAT
     ON FACTURAT.CodCl=INCASAT.CodCI ) TEMP1
  INNER JOIN
  (SELECT MAX (Vinzari - VALUE(Incasari, 0)) AS DifMax
FROM
  (SELECT CodCI, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS
  Vinzari FROM FACTURI F, LINIIFACT LF, PRODUSE P WHERE
  F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr GROUP BY CodCI) FACTURAT
  LEFT OUTER JOIN (SELECT CodCI, SUM(Transa) AS Incasari FROM
  FACTURI F, INCASFACT I WHERE I.NrFact=F.NrFact GROUP BY CodCI)
  ÎNCASAT
     ON FACTURAT.CodCl=INCASAT.CodCI) TEMP2
  ON TEMP1.DeIncasat=TEMP2.DifMax

  INNER JOIN CLIENŢI ON TEMP1.CodCl=CLIENTI.CodCI
```

* şi Oracle:

```
SELECT DenCl, Vinzari, Incasari, Delncasat FROM
  CLIENŢI,
  (SELECT FACTURAT.CodCI, Vinzari,
        NVL(Incasari, 0) AS Incasari,
        Vinzari - NVL(Incasari, 0) AS Delncasat
  FROM
      (SELECT CodCI, SUM(Cantitate * PretUnit *
         (1+ProcTVA)) AS Vinzari FROM FACTURI F, LINIIFACT LF,
      PRODUSE P WHERE F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr
      GROUP BY CodCI) FACTURAT,
      (SELECT CodCI, SUM(Transa) AS Incasari FROM FACTURI F,
      INCASFACT I WHERE I.NrFact=F.NrFact GROUP BY CodCI)
```

```
    ÎNCASAT
    WHERE FACTURAT.CodCl-INCASAT.CodCI (+)        )  TEMP1,
  (SELECT MAX (Vinzari - NVL(Incasari, 0)) AS DifMax FROM
     (SELECT CodCI, SUM(Cantitate * PretUnit *
        (1+ProcTVA)) AS Vinzari
       FROM FACTURI F, LINIIFACT LF, PRODUSE P WHERE
       F.NrFact=LF.NrFact AND LF.CodPr=P.CodPr GROUP BY
       CodCl) FACTURAT,
    (SELECT CodCl, SUM(Transa) AS Incasari FROM FACTURI F,
    INCASFACT I WHERE I.NrFact=F.NrFact GROUP BY CodCl)
    ÎNCASAT
    WHERE FACTURAT.CodCl=INCASAT.CodCl (+) ) TEMP2
WHERE TEMP1.DeIncasat=TEMP2.DifMax AND
    TEMP1.CodCl=CLIENTI.CodCl
```

6.3, Subconsultări scalare în clauza SELECT

Standardul SQL-92 face trimitere şi la interogările scalare ce pot fi definite ca fraze `SELECT` ce obţin un rezultat alcătuit dintr-o singură linie şi o singură coloană. Utilitatea acestora este vizibilă în expresii complexe. Deşi este prima oară când pomenim de interogări scalare, le-am folosit de multe ori până acum în clauzele `WHERE` şi `HAVING`. Ceea ce vom parcurge în continuare se referă la includerea unei interogări scalare în clauza `SELECT` a unei interogări. Din păcate, nici Oracle 8, nici VFP un au implementată această facilitate, aşa încât toate exemplele care urmează sunt specifice DB2.

Care sunt totalurile salariilor tarifare şi ale sporurilor pe luna iulie 2000 pentru întreaga firmă?

Soluţia clasică este:

```
SELECT SUM(SalTarifar) AS Total_Sal_Tarifar,
    SUM (
    VALUE(SporVechime,0) + VALUE(SporNoapte,0)+
        VALUE(SporCD,0)+VALUE(AlteSpor,0)
        ) AS Total_Sporuri_Iulie                                    -
FROM PERSONAL2
    INNER JOIN SPORURI ON PERSONAL2.Marca=SPORURI.Marca WHERE
An=2000 AND Luna=7
```

întrucât toate persoanele din tabela PERSONAL au lucrat în luna iulie 2000 (nu a plecat nici un angajat din organizaţie), se poate formula şi interogarea:

```
SELECT SUM (SalTarifar) Total_Sal__Tarifar,
    (SELECT SUM ( VALUE(SporVechime,0) +
        VALUE(SporNoapte,0)+ VALUE(SporCD,0)+
        VALUE(AlteSpor, 0))
    FROM SPORURI
    WHERE An=2000 AND Luna=7) AS Total_Sporuri Iulie FROM
PERSONAL2
```

A doua coloană a rezultatului este obţinută printr-o interogare scalară care operează oarecum

independent de fraza SELECT principală, furnizându-i însă o valoare.

Care sunt totalurile vânzărilor şi încasărilor?
Formulăm o variantă curioasă:

```
SELECT
    (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
       AS Vinzari FROM LINIIFACT LF, PRODUSE P WHERE
    LF.CodPr=P.CodPr) AS Facturat,
    (SELECT SUM (Transa)
    FROM INCASFACT) AS încasat FROM SYSIBM.SYSDUMMY1
```

Fraza SELECT conţine două subconsultări scalare: una care extrage totalul vânzărilor, cealaltă, totalul încasărilor. Deoarece în clauza FROM era nevoie de o tabelă cu o singură linie, s-a folosit pseudotabela SYS IBM. SYSDUMMY1 (echivalenta [pseudo]tabelei DUAL din Oracle) care are un rând şi un atribut.

Care este ziua în care s-au emis cele mai multe facturi?

```
SELECT DataFact, COUNT(*) AS Nr_Facturi,
    (SELECT MAX(NrF)
    FROM
       (SELECT COUNT(*) AS NrF FROM
       FACTURI
       GROUP BY DataFact) Tl ) AS NrMax FROM FACTURI
GROUP BY DataFact HAVING COUNT(*) >=
    (SELECT MAX(NrF)
    FROM
          (SELECT COUNT(*) AS NrF
          FROM FACTURI
          GROUP BY DataFact) T2 )
```

Valorile coloanelor Nr_Facturi şi NrMax ale rezultatului din figura 6.7 sunt furnizate de două subconsultări scalare, una care calculează numărul zilnic al facturilor, iar a doua, număr.:, maxim de facturi emise într-o zi.

DATAFACT	NR_FACTURI	NRMAX
2000-08-01	4	4
2000-08-07	4	4

Figura 6.7. Zilele în care s-au întocmit cele mai multe facturi

Care sunt localităţile în care se află sediul fiecărui client?
Revenim la un exemplu banal, rezolvat atât de simplu prin joncţiune sau subconsultare, pentru *i* demonstra cât de mult ne putem complica viaţa în SQL (deşi au fost exemple mai convingătoare. Ei bine, această problemă supărător de simplă poate fi rezolvată şi prin subconsultări scalare:

```
SELECT DenCl,
    (SELECT Loc FROM LOCALITATI
    WHERE CodPost=CLIENTI.CodPost)
FROM CLIENŢI
ORDER BY Loc
```

Figura 6.8 este edificatoare în ceea pc priveşte corectitudinea rezultatului interogării. Unicul merit al exemplului este, probabil, acela de a demonstra că o subconsultare scalară poate fi corelată cu tabela din clauza FROM a frazei principale.

DENCL	LOC
Clienţi SRL	Iasi
Client 2 SA	Iasi
Client 4	Paşcani
Client 6 SA	Roman
Client 5 SRL[1]	Timişoara
Client 7 SRL	Timişoara
Client 3 SRL	Vaslui

Figura 8.0. Subconsultare scalară corelată cu tabela din fraza principală

Continuăm cu alte exemple în care subconsultările scalare îşi arată pe deplin eficienţa. Abia o dată cu apariţia funcţiilor OLAP în SQL-99 şi versiunile recente ale Oracle (8i2), DB2 (6.1 şi 7) pot fi formulate soluţii bazate pe funcţii şi clauze preferabile subconsultărilor scalare.

Pentru cât la sută dintre clienţi s-au întocmit, zilnic, facturi?

```
SELECT DataFact, COUNT(DISTINCT CodCl) AS Nr Clienţi,
    (SELECT COUNT(*) FROM CLIENŢI) AS Nr Total Clienţi
    (COUNT(DISTINCT CodCl) * 100) /
       (SELECT COUNT(*) FROM CLIENŢI) AS Procent FROM
FACTURI GROUP BY DataFact
```

DATAFACT	NR_CLIENTI	NR_TOTAL_CLIENTI	PROCENT
2000-08-01	4	7	57
2000-08-02	2	7	28
2000-08-03	1	7	14
2000-08-04	1	7	14
2000-08-07	4	7	57

Figura 6.9. Procentajul zilnic al clienţilor pentru care există facturi

Pentru a obţine procentul care interesează, se împarte rezultatul calculat de funcţia COUNT pentru fiecare grup (zi calendaristică) la valoarea extrasă de interogarea scalară.

Care este contribuţia (procentuală) a fiecărui produs la totalul vânzărilor?

Informaţia este una esenţială pentru orice firmă. Ne vom servi de o subconsultare scalară pentru a determina, pe fiecare linie (corespunzătoare unui produs), totalul vânzărilor şi a face astfel raportul care ne interesează.

```
SELECT
   DenPr AS Produs,
   SUM(Cantitate * PretUnit * (1+ProcTVA))
       AS Vinzari_Produs,
   (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
   FROM LINIIFACT LF, PRODUSE P
   WHERE LF.CodPr=P.CodPr ) AS Total_Vinzari,
   (SUM(Cantitate * PretUnit * (1+ProcTVA)) * 100) / (SELECT
       SUM(Cantitate * PretUnit * (1+ProcTVA)) FROM LINIIFACT
       LF, PRODUSE P WHERE LF.CodPr=P.CodPr )
   AS Procent FROM LINIIFACT LF, PRODUSE P WHERE
LF.CodPr=P.CodPr GROUP BY DenPr
```

PRODUS	VINZARLPRODUS	T0TAL_VINZAR1	PROCENT
Produs 1	3385550.00	35839230.00	9
Produs 2	11356170.00	35839230.00	31
Produs 3	690200.00	35839230.00	1
Produs 4	1397060.00	35839230.00	3
Produs 5	19010250.00	35839230.00	53

Figura 6.10. Contribuţia fiecărui produs la cifra de afaceri

Care este evoluţia zilnică a vânzărilor, raportat la ziua calendaristică anterioară?

```
SELECT DataFact AS Zi,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari_Zi_Curenta,
   (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
   FROM FACTURI F2
      INNER JOIN LINIIFACT LF2 ON F2.NrFact=LF2.NrFact INNER
   JOIN PRODUSE P2 ON LF2.CodPr=P2.CodPr WHERE DataFact =
   F.DataFact - 1 DAY )
      AS Vinzari__Zi_Precedenta,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) -
      VALUE((SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)) FROM
            FACTURI F2
            INNER JOIN LINIIFACT LF2 ON
            F2.NrFact=LF2.NrFact INNER JOIN PRODUSE P2 ON
            LF2.CodPr=P2.CodPr WHERE DataFact = F.DataFact -
            1 DAY ),0)
      AS Diferenţa
   FROM FACTURI F INNER JOIN LINIIFACT LF ON F.Nrfact=LF.NrFact
   INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr
   GROUP BY DataFact
```

Soluţia este una mai laborioasă, dar şi mai de efect (figura 6.11). Prima interogare scalară furnizează totalul vânzărilor pentru ziua precedentă, ziua curentă fiind cea a grupului de referinţă. A doua instanţă a interogării scalare permite calcularea diferenţei dintre vânzările zilei curente şi a celei precedente. Funcţia VALUE converteşte valoarea NULL, datorată inexistenţei vânzărilor în data calendaristică precedentă (aşa cum este cazul zilelor de 1 şi 7 august 2000), în zero.

Zi	ViNZAR _ZI_CURENTA	VINZARI_ZI_PRECEDENTA	DIFERENŢA
2000-08-01	1 4684005.00		14684005.00'
2000-08-02	3034500.00	14684005.00	-1 1649505.00
2000-08-03	2320500.00	3034500.00	-714000.00
2000-08-04	2052750.00	2320500.00	-267750.00
2000-08-07	13747475.00		13747475.00

Figura 6.11. Diferenţa vânzărilor între ziua curentă şi cea precedentă

Care este evoluţia zilnică a vânzărilor, raportat la ziua de vânzări anterioară?

In problema de mai sus, diferenţa era calculată între ziua curentă şi ziua precedentă. Astfel încât toate zilele de luni erau raportate la zero, deoarece, pentru cea mai mare parte a firmelor, duminica nu se lucrează. Acum dorim ca diferenţa să fie calculată între vânzările din ziua curentă şi cele din *precedenta zi de vânzări,* adică între luni şi vineri ş.a.m.d., după cum se observă în figura 6.12.

ZI	VINZARLZLCURENTA	VIMZ^RI_ZI_PRECEDENTA	DIFERENŢA
2000-08-01	14684005.00		14684005.00
2000-08-02	3034500.00	14684005.00	-11649505.00
2000-08-03	2320500.00	3034500.00	-714000.00
2000-08-04	2052750.00	2320500.00	-267750.00
2000-08-07	13747475.00	2052750.00	11694725.00

Figura 6.12. Diferenţele dintre două zile consecutive de vânzări

```
SELECT DataFact AS Zi,SUM(Cantitate * PretUnit * (1+ProcTVA)) AS
        Vinzari_Zi_Curenta,
   (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
   FROM FACTURI F2
        INNER JOIN LINIIFACT LF2 ON F2.NrFact=LF2.NrFact
   INNER JOIN PRODUSE P2 ON LF2.CodPr=P2.CodPr WHERE
   DataFact IN
     (SELECT MAX(DataFact)
     FROM FACTURI F3
        INNER JOIN LINIIFACT LF3 ON F3.NrFact=LF3.NrFact
   INNER JOIN PRODUSE P3 ON LF3.CodPr=P3.CodPr WHERE
   DataFact < F.DataFact AND
        (Cantitate * PretUnit * (1+ProcTVA)) > 0)
   ) AS Vinzari_Zi_Precedenta,
                SUM(Cantitate * PretUnit * (1+ProcTVA)) - VALUE(
                   (SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
                   FROM FACTURI F2
                      INNER JOIN LINIIFACT LF2 ON F2.NrFact=LF2.NrFact
                   INNER JOIN PRODUSE P2 ON LF2.CodPr=P2.CodPr WHERE
                   DataFact IN
                      (SELECT MAX(DataFact)
                      FROM FACTURI F3                                    -
```

```
INNER JOIN LINIIFACT LF3 ON F3.NrFact=LF3.NrFact INNER JOIN PRODUSE P3 ON
    LF3.CodPr=P3.CodPr WHERE DataFact < F.DataFact AND
            (Cantitate * PretUnit * (1+ProcTVA)) > 0)
    ),0) AS Diferenţa
FROM FACTURI F
    INNER JOIN LINIIFACT LF ON F.Nrfact=LF.NrFact INNER
JOIN PRODUSE P ON LF.CodPr=P.CodPr GROUP BY DataFact
```

Artificiul necesar extragerii precedentei zi de vânzări se găseşte în subconsultările din cele două interogări scalare. Prin joncţiunea instanţelor F3 şi LF3 ale tabelelor FACTURI şi LINIIFACT şi condiţia DataFact < F.DataFact, vor fi extrase toate liniile din facturi întocmite înaintea datei curente (F este instanţa principală a tabelei FACTURI — cea care indică ziua curentă). Ca măsură suplimentară de precauţie se verifică dacă Cantitate * PretUnit * (1 + ProcTVA)) > 0 (ca nu cumva să fie vreo zi în care apare o factură în care liniile să prezinte Cantitate = 0, deşi situaţia aceasta este greu de imaginat). Dintre zilele de vânzare ce precedă ziua curentă, cea mai apropiată (calendaristic) se extrage prin funcţia MAX.

Nu ştiu ce ziceţi dumneavoastră, dar mie-mi place,..

6.4. Interogări ierarhice

Un titlu mai adecvat al acestui paragraf ar fi fost *Ierarhii în SQL* sau, şi mai general, *Structuri arborescente în SQL*. Cred că, dintre cei care au scris despre SQL, lider al subiectului este de departe Joe Celko, ce publică periodic un material pe această temă, fie în cărţile sale, fie în reviste precum *DBMS Magazine,* actualmente *Intelligent Enterprise.*

Preambulul discuţiei este plasat în paragraful 5.1, unde au fost prezentate două tabele, PERSONAL2 şi SPORURI. Prima dintre acestea reflectă structura ierarhică a firmei, deoarece, pentru fiecare angajat, este indicată şi marca şefului său direct (figura 5.2).

Cele câteva chestiuni care sunt descrise în continuare privesc modul în care pot fi făcute comparaţii, analize între angajaţi aflaţi pe anumite niveluri ierarhice, între angajaţi şi şefii lor direcţi (sau indirecţi).

în tabela PERSONAL2 angajatul cu marca 1 este chiar directorul general, pentru acesta valoarea atributului MarcaŞef fiind NULL.

Soluţia clasică - autojoncţiunea

Pentru aflarea majorităţii informaţiilor privitoare la ierarhia firmei, soluţia obişnuită în SQL constă în joncţionarea a două instanţe ale tabelei PERSONAL2 (PI şi P2) după condiţia PI.MarcaSef = P2.Marca.

Prin interogarea DB2 (valabilă şi în VFP):

```
SELECT *
FROM PERSONAL2 PI INNER JOIN PERS0NAL2 P2 ON
    PI.MarcaSef=P2.Marca
```

se obţine un rezultat de forma celui din figura 6.13.

MARCA INUMEPREN	DATANAST) COMPART	MARCASEF	SALTARIFARI	MARCA1 j NUMEPREN1	DATANAST1	COMPARŢ1	MAR CASEF1 j SALTARIFAR 1
2IANGAJAT 2	1977-10-11 | FINANCIAR	1	4500000	1 ANGAJAŢI	1962-07-01	DIRECŢIUNE	i 6000000
3 iANGAJAT 3	1962-08-22 jMARKETiNG	1	**4500000:**	1 [ANGAJAT 1	1962-07-01	DIRECŢIUNE	6000000
IOiANGAJAT 10	1972-01-29 IRESURSEUMANE	1	3700000=	1: ANGAJAT 1	1962-07-01	DIRECŢIUNE	6000000
41ANGAJAT 4	! FINANCIAR	2	3800000I	2 I ANGAJAT 2	1977-10-11	FINANCIAR	1 4500003
5; ANGAJAT 5	1965-04-30 IFINANCIAR	2	4200000'	2!ANGAJAT 2	1977-10-11	FINANCIAR	1 4500000
8: ANGAJAT 8	1960-12-31 : MARKETING	3	2S000G0:	3! ANGAJAT 3	1962-08-22	MARKETING	1 4500003
9! ANGAJAT 9	1976-02-28 'MARKETING	3	4100000:	3iANGAJAT 3	1962-08-22	MARKETING	1 4500000
6 i ANGAJAT 6	1965-11-09 IFINANCIAR	5	3500000:	5!ANGAJAT 5	1965-04-30	FINANCIAR	2 4200000
7! ANGAJAT 7	i FINANCIAR	5	2800000:	5! ANGAJAT 5	1965-04-30	FINANCIAR	2 4200000

254

Figura 6.13. Autojoncţiunea tabelei PERSONAL2

Primele şase coloane corespund primei instanţe, PI, în timp ce restul - celei de-a doua instanţe, P2. PI este legată de calitatea de subordonat, iar P2 de cea de şef. De aceea, este mai nimerită denumirea lui PI ca SUBORDONAŢI, iar a lui P2, ŞEFI.

Să luăm în discuţie câteva probleme.

*Cum se numeşte şeful Angajatului*7?

- Soluţie SQL-92/DB2/VFP:

```
SELECT SEFI.NumePren
FROM PERSONAL2 SUBORDONAŢI INNER JOIN PERSONAL2 SEFI ON
SUBORDONAŢI.MarcaSef = SEFI.Marca WHERE SUBORDONAŢI.NumePren = 'ANGAJAT
7'
```

Care sunt subordonaţii direcţi ai Angajatului 2?

- Soluţie SQL-92/DB2/VFP:

```
SELECT SUBORDONAŢI.NumePren
FROM PERSONAL2 SUBORDONAŢI INNER JOIN PERSONAL2 SEFI ON
SUBORDONAŢI.MarcaSef = SEFI.Marca WHERE SEFI.NumePren = 'ANGAJAT 2'
```

Fireşte, cele două probleme pot fi rezolvate şi prin subconsultări, astfel :

```
SELECT NumePren FROM PERSONAL2
WHERE Marca IN
   (SELECT MarcaSef FROM PERSONAL2
   WHERE NumePren = 'ANGAJAT 7')
```

respectiv:

```
   SELECT NumePren FROM PERSONAL2
WHERE MarcaSef IN (SELECT Marca FROM
                        PERSONAL2
   WHERE NumePren = 'ANGAJAT 2')
```

Ultimele două soluţii sunt valabile în aproape orice SGBD.

Câţi subordonaţi are fiecare angajat al firmei?
Soluţia:

```
SELECT SEFI.NumePren, COUNT(*) AS Nr_Subordonati FROM PERSONAL2 SUBORDONAŢI
INNER JOIN PERSONAL2 SEFI ON SUBORDONAŢI.MarcaSef = SEFI.Marca GROUP BY
SEFI.NumePren
```

extrage numai pe cei care au măcar un subordonat. Dacă dorim ca rezultatul să conţină toţi angajaţii în ordine alfabetică, trebuie folosită joncţiunea externă la dreapta şi modificat argumentul funcţiei COUNT:

```
SELECT SEFI.NumePren,
   COUNT(SUBORDONAŢI.MarcaSef) AS Nr_Subordonati FROM PERSONAL2 SUBORDONAŢI
RIGHT OUTER JOIN PERSONAL2 SEFI ON SUBORDONAŢI.MarcaSef = SEFI.Marca GROUP BY
SEFI.NumePren
```

Figura 6.14 conţine urmările acestei interogări.

NUMEPREN	NR SUBORDONAŢI
ANGAJAT 1	3
ANGAJAT 1 0	0
ANGAJAT 2	2
ANGAJAT 3	2
ANGAJAT 4	0
ANGAJAT 5	2
ANGAJAT 6	0
ANGAJAT 7	0
ANGAJAT 8	0
ANGAJAT 9	0

Figura 6.14. Numărul subordonaţilor pentru fiecare salariat

Care sunt subordonaţii subordonaţilor directorului general?

Din punctul de vedere al arborelui ce oglindeşte ierarhia firmei (figura 6.15), ne interesează „nepoţii rădăcinii". Interogarea trebuie să parcurgă trei niveluri ierarhice şi, în final, să extragă salariaţii cu mărcile 4, 5, 8 şi 9.

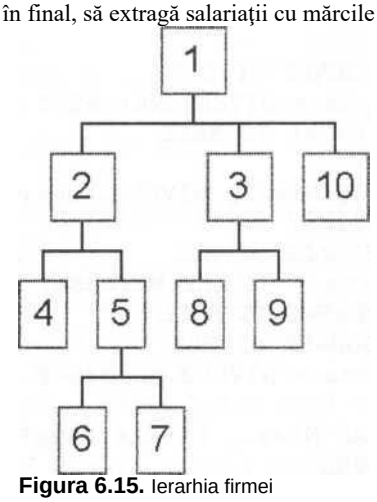


Figura 6.15. Ierarhia firmei

- Varianta 1 (generală):

```
SELECT SUBORDONAŢI.Marca,
SUBORDONAŢI.NumePren, SUBORDONAŢI.Compart FROM PERSONAL2 SUBORDONAŢI
INNER JOIN PERSONAL2 SEFI1 ON SUBORDONAŢI.MarcaSef = SEFI1.Marca INNER JOIN
PERSONAL2 SEFI2
      ON SEFI1.MarcaSef = SEFI2.Marca WHERE SEFI2.MarcaSef IS
NULL
```

- Varianta non-VFP (două niveluri de subconsultare):

```
SELECT Marca, NumePren, Compart FROM PERSONAL2
WHERE MarcaSef IN (SELECT Marca FROM PERSONAL2
WHERE MarcaSef IN (SELECT Marca FROM PERSONAL2
WHERE MarcaSef IS NULL)
   )
```

Care este nivelul ierarhic al fiecărui salariat?

De la început, ştim că ierarhia reflectată în tabela PERSONAL2 se derulează pe patru niveluri, aşa încât o interogare pentru rezolvarea problemei poate fi:

```
SELECT 'Nivel 1' AS Nivel, NumePren, Compart FROM PERSONAL2 WHERE
MarcaSef IS NULL UNION
SELECT 'Nivel 2' AS Nivel, NIVEL2.NumePren, NIVEL2.Compart
```

```
FROM PERSONAL2 NIVEL1
   INNER JOIN PERSONAL2 NIVEL2
      ON NIVEL1.Marca = NIVEL2.MarcaSef AND Nivell.MarcaSef IS NULL
   UNION
SELECT 'Nivel 3' AS Nivel, NIVEL3.NumePren, NIVEL3.Compart FROM PERSONAL2 NIVEL1
   INNER JOIN PERSONAL2 NIVEL2
      ON NIVEL1.Marca = NIVEL2.MarcaSef AND Nivell.MarcaSef IS NULL
   INNER JOIN PERSONAL2 NIVEL3
      ON NIVEL2.Marca = NIVEL3.MarcaSef UNION
SELECT 'Nivel 4' AS Nivel, NIVEL4.NumePren, NIVEL4.Compart FROM PERSONAL2 NIVEL1
   INNER JOIN PERSONAL2 NIVEL2
      ON NIVEL1.Marca = NIVEL2.MarcaSef AND Nivell.MarcaSef IS NULL
   INNER JOIN PERSONAL2 NIVEL3
      ON NIVEL2.Marca = NIVEL3.MarcaSef INNER JOIN PERSONAL2
   NIVEL4                                    •
      ON NIVEL3.Marca = NIVEL4.MarcaSef ORDER BY NumePren
```

Iată şi rezultatul.

NUMEPREN	COMPART	NIVEL
ANGAJAT 1	DIRECŢIUNE	Nivel 1
ANGAJAT 1 0	RESURSE UMANE	Nivel 2
ANGAJAT 2	FINANCIAR	Nivel 2
ANGAJAT 3	MARKETING	Nivel 2
^ANGAJAT 4	FINANCIAR	Nivel 3
ANGAJAT 5	FINANCIAR	Nivel 3
ANGAJAT 6	FINANCIAR	Nivel 4
ANGAJAT 7	FINANCIAR	Nivel 4
ANGAJAT 8	WRKETING	Nivel 3
ANGAJAT 9	MARKETING	Nivel 3

Figura 6.16. Nivelul ierarhic al fiecărui angajat

Pentru Visual FoxPro, diferenţa principală ţine de regimul clauzei ORDER BY. Aceasta trebuie să fie ataşată primei fraze SELECT din reuniune. în plus, paradoxal, coloana de ordonare poate fi indicată numai prin număr (în această situaţie). SELECT-ul care obţine rezultatul din figura 6.16 este:

```
SELECT 'Nivel 1' AS Nivel, NumePren, Compart ;
FROM PERSONAL2 ;
WHERE MarcaSef IS NULL ;
ORDER BY 2 ;
    UNION ;
SELECT 'Nivel 2' AS Nivel, NIVEL2.NumePren, NIVEL2.Compart ;
FROM PERSONAL2 NIVEL1 ;
   INNER JOIN PERSONAL2 NIVEL2 ;
      ON NIVEL1.Marca = NIVEL2.MarcaSef AND ;
         Nivell.MarcaSef IS NULL ;
   UNION ;
SELECT 'Nivel 3' AS Nivel, NIVEL3.NumePren, NIVEL3.Compart ; FROM PERSONAL2 NIVEL1 ;
   INNER JOIN PERS0NAL2 NIVEL2 ;
      ON NIVEL1.Marca = NIVEL2.MarcaSef AND ;
         Nivell.MarcaSef IS NULL ;
   INNER JOIN PERSONAL2 NIVEL3 ;
      ON NIVEL2.Marca = NIVEL3.MarcaSef ;
   UNION ;
SELECT 'Nivel 4' AS Nivel, NIVEL4.NumePren, NIVEL4.Compart ; FROM PERSONAL2 NIVEL1 ;
   INNER JOIN PERS0NAL2 NIVEL2 ;
      ON NIVEL1.Marca = NIVEL2.MarcaSef AND ;
         Nivell.MarcaSef IS NULL ;
```

```
    INNER JOIN PERSONAL2 NIVEL3 ;
        ON NIVEL2.Marca = NIVEL3.MarcaSef ;
    INNER JOIN PERS0NAL2 NIVEL4 ;
        ON NIVEL3.Marca = NIVEL4.MarcaSef
```

Interogarea funcţionează rezonabil. Există însă cel puţin două umbre: trebuie să ştim, aprioric, numărul nivelurilor ierarhice, iar în al doilea rând, la un număr mult mai mare de niveluri, întinderea consultării creşte sensibil.
O variantă mai elegantă, ca volum de scris, nu şi ca resurse consumate, este:

```
SELECT NIVEL4.*,
    CASE
        WHEN NIVEL4.MarcaSef      ISNULL THEN 1
        WHEN NIVEL3.MarcaSef      ISNULL THEN 2
        WHEN NIVEL2.MarcaSef      ISNULL THEN 3
        WHEN NIVEL1.MarcaSef      ISNULL THEN 4
    END AS Nivel FROM PERSONAL2 NIVEL1
    RIGHT OUTER JOIN PERSONAL2 NIVEL2
        ON NIVEL1.Marca = NIVEL2.MarcaSef AND Nivell.MarcaSef IS NULL
    RIGHT OUTER JOIN PERSONAL2 NIVEL3 ON NIVEL2.Marca =
    NIVEL3.MarcaSef RIGHT OUTER JOIN PERSONAL2 NIVEL4 ON
    NIVEL3.Marca = NIVEL4.MarcaSef
```

Se joncţionează extern patru instanţe ale tabelei PERSONAL2, câte una pentru fiecare nivel ierarhic, astfel încât au şi fost denumite NI VEL 1... NIVEL4.

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIFAR	NIVEL
11	ANGAJAT 1	1962-07-01	DIRECŢIUNE		6000000	1
10	ANGAJAT 10	1972-01-29	RESURSE UMANE	1	3700000	2
2	ANGAJAT 2	1977-10-11	FINANCIAR	1	4500000	2
3	ANGAJAT 3	1962-08-22	MARKETING	1	4500000	2
4	ANGAJAT 4		FINANCIAR	2	3800000	3
5	ANGAJAT 5	1965-04-30	FINANCIAR	2	4200000	3
6	ANGAJAT 6	1965-11-09	FINANCIAR	5	3500000	4
7	ANGAJAT 7		FINANCIAR	5	2800000	4
8	ANGAJAT 8	1960-12-31	MARKETING	3	2900000	3
9	ANGAJAT S	i 976-02-28	MARKETING	3	4100000	3

Figura 8.17. Soluţie DB2 pentru aflarea nivelului ierarhic al fiecărui angajat

Fraza SELECT compune ierarhia pornind de la NIVEL 1, care reprezintă o instanţă a PERSONAL2 cu o singură linie, cea a directorului general, instanţă joncţionată extern la dreapta cu NIVEL2, care va furniza subordonaţii direcţi ai înregistrării-rădăcină ş.a.m.d.

Transcrisă în dialectul VFP, interogarea se prezintă astfel:

```
SELECT NIVEL4.*,    ;
IIF(ISNULL(NIVEL4.MarcaSef), 1,         ;
      IIF(ISNULL(NIVEL3.MarcaSef), 2,         ;
         IIF(ISNULL(NIVEL2.MarcaSef), 3, 4) ) ) AS Nivel ; FROM PERSONAL2 NIVEL1 ;
   RIGHT OUTER JOIN PERSONAL2 NIVEL2 ;
      ON NIVEL1.Marca = NIVEL2.MarcaSef AND ;
         Nivell.MarcaSef IS NULL ;
   RIGHT OUTER JOIN PERSONAL2 NIVEL3 ;
      ON NIVEL2.Marca = NIVEL3.MarcaSef ;
   RIGHT OUTER JOIN PERSONAL2 NIVEL4 ;
      ON NIVEL3.Marca = NIVEL4.MarcaSef ;
```

Să se afişeze structura ierarhică a firmei.

Practic, dorim o formă de vizualizare a angajaţilor care să ţină cont de modul lor de subordonare - ca în figura 6.18.

NUME	COMPART	SEF1	SEF2 I	SEF3	SEF4
ANGAJAŢI	DIRECŢIUNE	1	1	1	1
-------ANGAJAT 2	FINANCIAR	1	2	2	2
---------- ANGAJAT 4	FINANCIAR	1	2	4	4
---------- ANGAJAT 5	FINANCIAR	1	2	5	5
.....ANGAJAT 6	FINANCIAR	1	2	5	6
----------ANGAJAT 7	FINANCIAR	1	2	5	7
-------ANGAJAT 3	MARKETING	1	3	3	3
---------- ANGAJAT 8	MARKETING	1	3	8	8
---------- ANGAJAT 9	MARKETING	1	3	9	9
-------ANGAJAT 1 0	RESURSE UMANE		10	10	10

Figura 6.18. Vizualizarea ierarhiei

Dintre variantele DB2 operaţionale, începem cu una destul de stufoasă, bazată pe expresia tabelă NIVELE.

```
WITH NIVELE AS
    (SELECT NIVEL4.*,
    CASE
        WHEN NIVEL4.MarcaSef        ISNULL  THEN   1
        WHEN NIVEL3.MarcaSef        ISNULL  THEN   2
        WHEN NIVEL2.MarcaSef        ISNULL  THEN   3
        WHEN NIVEL1.MarcaSef        ISNULL  THEN   4
    END AS Nivel FROM PERSONAL2 NIVEL1
    RIGHT OUTER JOIN PERSONAL2 NIVEL2 ON NIVEL1.Marca = NIVEL2.MarcaSef AND Nivell.MarcaSef IS
NULL RIGHT OUTER JOIN PERSONAL2 NIVEL3 ON NIVEL2.Marca = NIVEL3.MarcaSef RIGHT OUTER JOIN
PERSONAL2 NIVEL4 ON NIVEL3.Marca = NIVEL4.MarcaSef )
SELECT NIVELE.NumePren AS Nume, NIVELE.Compart,
    NIVELE.Marca AS Sefl, NIVELE.Marca AS Sef2,
    NIVELE.Marca AS Sef3, NIVELE.Marca AS Sef4 FROM NIVELE
WHERE Nivel = 1                                                    ■
    UNION
SELECT '----------'||N2.NumePren AS Nume, N2.Compart,
    NI.Marca AS Sefl, N2.Marca AS Sef2, N2.Marca AS Sef3,
    N2.Marca AS Sef4 FROM NIVELE N2 INNER JOIN NIVELE NI ON
    N2.Nivel=2 AND N2.MarcaSef=Nl.Marca AND Nl.Nivel=l UNION
SELECT '---------- _          '||N3.NumePren AS Nume, N3.Compart,
    NI.Marca AS Sefl, N2.Marca AS Sef2, N3.Marca AS Sef3, N3.Marca AS Sef4 FROM NIVELE N3
    INNER JOIN NIVELE N2
    ON N3.Nivel=3 AND N3.MarcaSef=N2.Marca AND N2.Nivel=2 INNER JOIN NIVELE NI
    ON N2.Nivel=2 AND N2.MarcaSef=Nl.Marca AND Nl.Nivel=l UNION
SELECT '----------' | |N4.NumePren AS Nume, N4.Compart,
    NI.Marca AS Sefl,
    N2.Marca AS Sef2, N3.Marca AS Sef3, N4.Marca AS Sef4 FROM NIVELE N4 INNER JOIN NIVELE N3
    ON N4.Nivel=4 AND N4.MarcaSef=N3.Marca AND N3.Nivel=3 INNER JOIN NIVELE N2
    ON N3.Nivel=3 AND N3.MarcaSef=N2.Marca AND N2.Nivel=2 INNER JOIN NIVELE NI
    ON N2.Nivel=2 AND N2.MarcaSef=N1.Marca AND Nl.Nivel=l ORDER BY Sefl. Sef2, Sef3. Sef4
```

Lucrurile pot fi simplificate sensibil utilizând o structură de tip CASE, prin care determinăm nivelul de subordonare, nivel care va determina de câte ori se repetă (funcţia DB2 REPEAT) grupul de şase liniuţe, astfel încât listarea primei coloane este asemănătoare celei din figura 6.18.

```
SELECT RTRIM ( CHAR ( REPEAT                    6 * (       (
    CASE
        WHEN      NIVEL4.MarcaSef      IS NULL   THEN  1
        WHEN      NIVEL3.MarcaSef      IS NULL   THEN  2
        WHEN      NIVEL2.MarcaSef      IS NULL   THEN  3
        WHEN      NIVEL1.MarcaSef      IS NULL   THEN  4
    END ) - 1 )          )     ) )    M
    NIVEL4.NumePren AS Nume, NIVEL4.Compart FROM PERSONAL2 NIVEL1
    RIGHT OUTER JOIN PERSONAL2 NIVEL2 ON NIVEL1.Marca =
    NIVEL2.MarcaSef AND Nivel1.MarcaSef IS NULL RIGHT OUTER JOIN
    PERSONAL2 NIVEL3
            ON NIVEL2.Marca = NIVEL3.MarcaSef RIGHT OUTER JOIN PERSONAL2 NIVEL4
            ON NIVEL3.Marca = NIVEL4.MarcaSef
```

In VFP, acelaşi rezultat se obţine printr-un lanţ de IF-uri imediate (IIF-uri):

```
SELECT R E P L I C A T E ,      (IIF(ISNULL(NIVEL4.MarcaSef), 1, ;
        IIF(ISNULL(NIVEL3.MarcaSef), 2, ;
            IIF(ISNULL(NIVEL2.MarcaSef), 3, 4) ) ) - 1) ;
        *   7)+NIVEL4.NumePren AS Nume, NIVEL4.Compart ;
FROM PERSONAL2 NIVEL1 ;
    RIGHT OUTER JOIN PERSONAL2 NIVEL2 ;
        ON NIVEL1.Marca = NIVEL2.MarcaSef AND ;
            Nivell.MarcaSef IS NULL ;
    RIGHT OUTER JOIN PERSONAL2 NIVEL3 ;
        ON NIVEL2.Marca = NIVEL3.MarcaSef ;
    RIGHT OUTER JOIN PERSONAL2 NIVEL4 ;
        ON NIVEL3.Marca = NIVEL4.MarcaSef
```

Interogări arborescente în Oracle

Pentru problemele formulate în acest paragraf, până acum au fost formulate numai soluţii DB2 şi VFP, lăsând să se înţeleagă că formularea variantelor echivalente în Oracle ar fi o temă pentru acasă/la birou. De fapt ne-am rezervat pentru paginile care urmează. Dialectul SQL al Oracle are câteva opţiuni speciale pentru parcurgerea structurilor ierarhice, cele mai importante fiind START WITH şi CONNECT BY.

Care este nivelul ierarhic al fiecărui salariat?

Soluţia următoare conduce la rezultatul din figura 6.19.

```
SELECT PERSONAL2.*, LEVEL FROM PERSONAL2 START WITH
MarcaSef IS NULL CONNECT BY PRIOR Marca=MarcaSef
```

Construirea structurii ierarhice începe cu înregistrarea (înregistrările) care îndeplineşte (îndeplinesc) condiţia din clauza START WITH. Această înregistrare-părinte va fi legată de înregistrarea sau înregistrările-copil prin condiţia Marca = MarcaSef. Clauza PRIOR plasată în stânga condiţiei semnifică: valoarea atributului Marca din părinte trebuie să fie egală cu valoarea MarcaSef din înregistrările-copil.

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIFAR	LEVEL
1	ANGAJAT 1	01 -JUL-62	DIRECŢIUNE		6000000	1
2	ANGAJAT 2	11-OCT-77	FINANCIAR	1	4500000	2
4	ANGAJAT 4		FINANCIAR	2	3800000	3
5	ANGAJAT 5	30-APR-65	FINANCIAR	2	4200000	3
6	ANGAJAT 6	09-NOV-65	FINANCIAR	5	3500000	4
7	ANGAJAT 7		FINANCIAR	5	2800000	.. 4
3	ANGAJAT 3	22-AUG-62	MARKETING	1	4500000	2
8	ANGAJAT 8	31-DEC-60	MARKETING	3	2900000	3
9	ANGAJAT 9	28-FEB-76	MARKETING	3	4100000	3
10	ANGAJAT 10	29-JAN-72	RESURSE UMANE	1	5500000	2

Figura 6.19. Interogare ierarhică

Prin CONNECT BY sunt selectate toate generaţiile succesive de linii-copil (copii, nepoţi, strănepoţi etc.). După construirea ierarhiei, se elimină tuplurile ce nu îndeplinesc condiţia formulată în clauza WHERE. Este important de notat că selecţia se aplică linie cu linie, iar eliminarea unei linii-părinte nu atrage automat eliminarea copiilor, nepoţilor ş.a.m.d. Dacă există clauza ORDER BY, aceasta va determina dispunerea înregistrărilor în rezultat. In lipsa clauzei de ordonare, înregistrările sunt dispuse în funcţie de ordinea parcurgerii arborelui, aşa cum se observă din figura 6.15.

Un avantaj major al interogărilor ierarhice ţine de folosirea pseudocoloanei LEVEL, ce semnifică tocmai nivelul ierarhiei, relativ la înregistrarea/înregistrările-„rădăcină" care îndeplineşte/îndeplinesc condiţia din START WITH.

Ca principale restricţii trebuie amintit că SELECT-ul care execută o interogare ierarhică nu poate efectua o joncţiune şi nici extrage date dintr-o tabelă virtuală creată printr-o joncţiune.

Cum se numeşte şeful Angajatului 7?

Dacă folosim interogarea:

```
SELECT PERSONAL2.*, LEVEL
FROM PERSONAL2
START WITH NumePren = 'ANGAJAT 7'
CONNECT BY PRIOR MarcaSef = Marca
```

se obţine:

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIFAR	LEVEL
7	ANGAJAT 7		FINANCIAR	5	2800000	1
5	ANGAJAT 5	30-APR-65	FINANCIAR	2	4200000	2
2	ANGAJAT 2	11-OCT-77	FINANCIAR	1	4500000	3
1 ANGAJAT 1		01-JUL-62	DIRECŢIUNE		6000000	4

Răspunsul exact la întrebare (figura 6.21) presupune următoarea soluţie:

```
SELECT PERSONAL2.*, LEVEL FROM PERSONAL2 WHERE LEVEL_- 1 =
    ( SELECT LEVEL FROM PERSONAL2
    WHERE NumePren = 'ANGAJAT 7'
    START    WITH          NumePren = 'ANGAJAT 7'
    CONNECT BY  PRIOR MarcaSef = Marca                )
START WITH NumePren = 'ANGAJAT 7'
CONNECT BY PRIOR MarcaSef = Marca
```

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIFAR	LEVEL
5	ANGAJAT 5	30-APR-65	FINANCIAR	2	4200000	2

Figura 6.21. Şeful direct al Angajatului 7

Care sunt subordonaţii direcţi ai Angajatului 2?

Subordonaţii de ordin 1 (direcţi), 2, 3..., adică cei din figura 6.22, pot fi extraşi prin:

```
SELECT PERSONAL2.*, LEVEL FROM PERSONAL2
START WITH NumePren = 'ANGAJAT 2'
CONNECT BY MarcaSef = PRIOR Marca
```

MARCA	NUMEPREN	DATANAST	COMPART	MARCASEF	SALTARIFAR	LEVEL
2	ANGAJAT 2	11-OCT-77	FINANCIAR 1		4500000	h
4	ANGAJAT 4		FINANCIAR 2		3800000	2
5	ANGAJAT 5	30-APR-65	FINANCIAR 2		4200000	2
6	ANGAJAT 6	09-NOV-65	FINANCIAR 5		3500000	3
7	ANGAJAT 7		FINANCIAR 5		2800000	3

Figura 6.22. Toţi subordonaţii Angajatului 2

Ca să obţinem răspunsul punctual la întrebare, apelăm la o subconsultare:

```
SELECT PERSONAL2.*, LEVEL FROM PERSONAL2 WHERE LEVEL - 1 =
    (SELECT LEVEL
    FROM PERSONAL2
    WHERE NumePren = 'ANGAJAT 2'
    START WITH NumePren = 'ANGAJAT 2'
    CONNECT BY PRIOR MarcaSef = Marca)
START WITH NumePren = 'ANGAJAT 2'
CONNECT BY MarcaSef = PRIOR Marca
```


Care sunt subordonaţii subordonaţilor directorului general?

Acestea sunt înregistrările-„nepot" ale înregistrării-rădăcină (MarcaSef IS NULL)

O soluţie bazată tot pe pseudocoloana LEVEL şi o subconsultare este:

islll

pHii-.1:

:

```
SELECT PERS0NAL2.*, LEVEL
FROM PERSONAL2 WHERE
LEVEL - 2 =
   (SELECT LEVEL
   FROM PERSONAL2
   WHERE MarcaSef IS NULL
   START WITH MarcaSef IS NULL
   CONNECT BY PRIOR MarcaSef = Marca )
START WITH MarcaSef IS NULL CONNECT
BY MarcaSef = PRIOR Marca
```

însă dacă ţinem seama că nivelul ierarhic al directorului general este 1, al subordonaţilor săi direcţi este 2, iar al subordonaţilor subordonaţilor este 3, soluţia se simplifică apreciabil:

```
SELECT PERSONAL2.*, LEVEL
FROM PERSONAL2
WHERE LEVEL = 3
START WITH  MarcaSef  IS  NULL
CONNECT BY  MarcaSef  = PRIOR    Marca
```

Să se afişeze structura ierarhică a firmei.

Nu putem să reprezentăm arborele ierarhic cu „verdele în jos", ci cu rădăcina la stânga, la fel ca în soluţia DB2 de acum câteva figuri. Indentarea subordonaţilor se obţine cu ajutorul funcţiei LPAD şi psedoatributului LEVEL după cum urmează:

```
SELECT LPAD('    ',   5*(LEVEL - 1),              || numepren AS Nume,
   Compart FROM PERS0NAL2 START WITH
              MarcaSef  IS  NULL
CONNECT BY  PRIOR Marca  = MarcaSef
```

în lipsa unei condiţii formulate în clauza START WITH, se construieşte câte o ierarhie pentru fiecare angajat (fiecare angajat va fi, pe rând, rădăcină a unei ierarhii). Astfel, interogarea următoare va genera rezultatul din figura 6.24.

Figura 6.20. Toţi şefii Angajatului 7

NUME	COMPART
ANGAJAT 1	DIRECŢIUNE
— ANGAJAT 2	FINANCIAR
- ..... ANGAJAT 4	FINANCIAR
..... ANGAJAT 5	FINANCIAR
- .....ANGAJAT 6	FINANCIAR
— .....ANGAJAT 7	FINANCIAR
—- ANGAJAT 3	MARKETING
----------ANGAJAT 8	MARKETING
— ..... ANGAJAT 9	MARKETING
— ANGAJAT 10	RESURSE UMANE

NUME	COMPART
ANGAJAT 1	DIRECŢIUNE
.... ANGAJAT 2	FINANCIAR
.....— ANGAJAT *4*	FINANCIAR
.....— ANGAJAT 5	FINANCIAR
.....- ANGAJAT 6	FINANCIAR
..... ANGAJAT 7	FINANCIAR
.... ANGAJAT 3	MARKETING
..... ANGAJAT 8	MARKETING
..... ANGAJAT 9	MARKETING
.... ANGAJAT 10	RESURSE UMANE
ANGAJAT 2	FINANCIAR
— ANGAJAT 4	FINANCIAR
.... ANGAJAT 5	FINANCIAR
- .....- ANGAJAT 6	FINANCIAR
..... - ANGAJAT 7	FINANCIAR
ANGAJAT 3	MARKETING
— ANGAJAT 8	MARKETING
.... ANGAJAT 9	MARKETING
ANGAJAT 4	FINANCIAR
ANGAJAT 5	FINANCIAR
.... ANGAJAT 6	FINANCIAR
— ANGAJAT 7	FINANCIAR
ANGAJAT 6	FINANCIAR
ANGAJAT 7	FINANCIAR
ANGAJAT 8	MARKETING
ANGAJAT 9	MARKETING
ANGAJAT 10	RESURSE UMANE

Figura 6.24. Ierarhii pentru fiecare angajat

```
SELECT LPAD('      5*(LEVEL - 1),          || numepren AS Nume,
                Compart FROM PERSONAL2
CONNECT BY PRIOR Marca = MarcaSef
```

Primele zece înregistrări reprezintă structura ierarhică pentru care rădăcina este directorul general; liniile 11 -15 ale rezultatului constituie ierarhia a cărei bază este *Angajat 2* ş.a.m.d.

6.5. Actualizarea tabelelor prin subconsultări

Pentru o mai bună priză la public, în cele ce urmează apelăm la ceea ce se numeşte de-normalizarea

bazei de date. în practică, redundanţa datelor este uneori condamnată cu jumătate de gură sau chiar apreciată de mulţi specialişti. Aceasta deoarece un atribut redundant, adăugat unei tabele, poate duce la evitarea joncţiunilor frecvente, mari consumatoare de resurse. Aplecându-ne asupra bazei noastre de date, dacă tot ne apucăm de treabă, în tabela FACTURI adăugăm nu mai puţin de trei atribute:

- ValTotală reprezintă valoarea totală (inclusiv TVA) a facturii;
- Reduceri: într-o ţară civilizată, ca a noastră, dacă un client plăteşte o factură înainte de termenul obişnuit (să zicem zece zile), îi putem acorda o reducere de 5% pentru că ne-am procopsit rapid cu lichidităţi.
- Penalităţi: este opusul atributului anterior. Prin contract sau prin lege, dacă un client este mai reţinut în a plăti facturile la timp, i se pot aplica penalităţi, fară a avea însă certitudinea că le-am putea încasa vreodată.

Practic, prin aceste noi atribute, urmărirea încasării facturilor se schimbă sensibil. Valoarea de încasat dintr-o factură este valoarea totală plus penalităţi minus reduceri. Asta e vestea bună. Vestea proastă ţine de faptul că atributul ValTotală n-ar trebui să se modifice decât la actualizarea tabelei LINIIFACT, nu? Altminteri, dacă utilizatorul ar modifica, prin UPDATE sau alt mijloc, acest atribut, ar apărea un decalaj supărător între liniile din facturi şi valorile acestora. Actualizarea automată a unor atribute calculate este unul din scopurile declarate ale declanşatoarelor *(trigger-e*lor), după cum vom vedea în capitolul 8.

Deocamdată adăugăm cele trei atribute tabelei FACTURI:

```
ALTER TABLE FACTURI ADD ValTotala
   DECIMAL(16) ;
ALTER TABLE FACTURI ADD Reduceri
   DECIMAL(15) ;
ALTER TABLE FACTURI ADD Penalizari
   DECIMAL(15) ;
```

Acum, dacă tot le-am creat, să le „umplem". Astfel, pentru calculul valorii totale a unei facturi avem nevoie de o interogare corelată după cum urmează:

```
UPDATE FACTURI
   SET ValTotala = (
      SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
      FROM LINIIFACT LF INNER JOIN PRODUSE P ON
      LF.CodPr=P.CodPr WHERE NrFact = FACTURI.NrFact )
```

Subconsultarea ia în calcul cantitatea, preţul unitar şi procentul TVA pentru fiecare produs vândut. Prin corelare se vor lua în considerare numai liniile din LINIIFACT (şi PRODUSE) corespunzătoare facturii curente (linia curentă din FACTURI).

In Oracle, singura deosebire ţine de exprimarea joncţiunii interne:

```
UPDATE FACTURI
   SET ValTotala = (
      SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
      FROM LINIIFACT LF, PRODUSE P
      WHERE LF.CodPr=P.CodPr AND NrFact = FACTURI.NrFact
      )
```

Cu regretele de rigoare, trebuie să constatăm că şi această interogare este prea savantă pentru dialectul SQL din VFP. Astfel, toate comenzile de actualizare următoare vor fi tratate cu ostilitate de acest SGBD.

Cât priveşte atributul Reduceri, instituim următoarea regulă: se acordă o reducere de 5% pentru toate tranşele unei facturi încasate în termen de 15 zile de la data vânzării.

```
UPDATE FACTURI
    SET Reduceri = (
        SELECT SUM ( CASE
            WHEN Datainc <= DataFact + 15 DAY THEN Transa
            * 0.05 ELSE 0 END)
        FROM INCASFACT INCF
            INNER JOIN INCASARI I
                ON INCF.CodInc=I.Codlnc INNER
            JOIN FACTURI F2
                ON INCF.Nrfact=F2.NrFact WHERE
        F2.NrFact- = FACTURI.NrFact )
```

Noul conţinut al tabelei FACTURI este cel din figura 6.25.

NRFACT	DATAFACT	CODCL	OBS	VALTOTALA	REDUCERI	PENALIZARI
1111	2000-08-01	1001		5399625	269981	
1112	2000-08-01	1005	Probleme...	1337560	24385	
1113	2000-08-01	1002		1160250	0	
1114	2000-08-01	1006		6786570		
1115	2000-08-02	1001		1651125		
1116	2000-08-02	1007	Preţul prop...	1383375		
1117	2000-08-03	1001		2320500	116025	
1118	2000-08-04	1001		2052750	102637	
1119	2000-08-07	1003		7242935		
1120	2000-08-07	1001		1066240[1]	36577	
1121	2000-08-07	1004		5438300		
1122	2000-08-07	1005		0		

Figura 6.25. Reduceri de 5% pentru încasări în mai puţin de 15 zile de la facturare

Interogarea Oracle 8i are două diferenţe: una legată de joncţiune, iar cealaltă, de modul de redactare a expresiei de tip dată calendaristică.

```
UPDATE FACTURI
    SET Reduceri = (
        SELECT SUM ( CASE
            WHEN Datainc <= DataFact + 15 THEN Transa * 0.05
                ELSE 0 END)
        FROM INCASFACT INCF, INCASARI I, FACTURI F2
        WHERE INCF.CodInc=I.Codlnc AND
            INCF.Nrfact=F2.NrFact AND
            F2.NrFact = FACTURI.NrFact
        )
```

Complicăm un pic cazul. Acordăm 10% pentru tranşele încasate în mai puţin de 15 zile de la data vânzării, 9% pentru 16 zile şi 8% pentru 17 zile. Soluţia DB2 este:

```
UPDATE FACTURI
    SET Reduceri = (
        SELECT SUM ( CASE
            WHEN Datalnc <= DataFact + 15 DAY
            THEN Transa * 0.1 WHEN Datalnc <=
            DataFact + 16 DAY THEN Transa * 0.09
            WHEN Datalnc <= DataFact + 17 DAY
```

```
    THEN Transa * 0.08 ELSE 0 END)
FROM INCASFACT INCF
    INNER JOIN INCASARI I
        ON INCF.CodInc=I.Codlnc INNER JOIN FACTURI F2 ON
INCF.Nrfact=F2.NrFact WHERE F2.NrFact = FACTURI.NrFact )
```

NRFACT	DATAFACT	CODCL	OBS	VALTOTALA	REDUCERI	PENAUZARI
1111	2000-08-01	1001		5399625	539962	
1112	2000-08-01	1005	Probleme...	1337560		
, 1113	2000-08-01	1002		1160250	104422	
1114	2000-08-01	1006		6786570		
1115	2000-08-02	1001		^ 1651125		
1116	2000-08-02	1007	Preţul prop...	1383375		
1117	2000-08-03	1001		2320500	232050	
1118	2000-08-04	1001		2052750	205275	
1119	2000-08-07	1003		7242935		
1120	2000-08-07	1001		1066240	73155	
1121	2000-08-07	1004		5438300		
1122	2000-08-07	1005		0		

Figura 6.26. Reduceri pe tranşe

Nu numai modificările pot fi operate utilizând subconsultări, corelate sau nu, ci şi ştergerile. Spre exemplu, vrem să ştergem din tabela FACTURI liniile care nu au nici un copil în LINIIFACT:

```
DELETE FROM
FACTURI WHERE NOT
EXISTS (SELECT 1
FROM LINIIFACT
    WHERE LINIIFACT.NrFact = FACTURI.NrFact)
```

De această dată, şi VFP este mai conciliant, lucrurile petrecându-se identic ca în DB2 şi Oracle. O soluţie mai puţin impresionantă utilizează tot o subconsultare, dar ceva mai puţin corelată:

```
DELETE FROM
FACTURI WHERE NrFact
NOT IN
    (SELECT DISTINCT NrFact
    FROM LINIIFACT)
```

Şi această soluţie este una generală (DB2/Oracle/VFP).

Caitolul 7 SQL ŞI OLAP

Cunoaştem cu toţii zicala cu fratele, dracul şi puntea[30]. O folosim destui, mai ales în momentele de compromis, fireşte, flecare imaginându-şi că celălalt e dracul. în materie de funcţii analitice, care au fost introduse ca amendament la standardul SQL-99, puntea este ANSI (American National Standard

30 Nu e prea elegant să te autocitezi sau să repeţi glumele, mai ales atunci când sunt slabe. însă, pur şi simplu, nu m-am putut abţine să nu inserez introducerea din materialul publicat în *PC Report*, nr. 98 (noiembrie)/2000. Vezi [Fotache00-4] şi [Fotache00-5].

Institute), iar cei doi parteneri principali (care o fi necuratul?) sunt IBM şi Oracle.

Derulată de pe la începutul anului 1999, colaborarea dintre echipele conduse de Hamid Paresh la IBM (Almaden Research Laboratory) şi Andy Witkowski la Oracle s-a concretizat rapid într-un set comun de specificaţii pe care l-au înaintat ANSI (de la ANSI, Jim Melton a jucat un rol important în acest proiect). în scurt timp, pe baza specificaţiilor, ANSI a publicat *Amendamentul 1* la standardul SQL-99.

IBM a implementat o parte din opţiunile OLAP ale SQL în versiunea 6.1 a DB2-ului, în timp ce Oracle în versiunea 8i2, ambele lansate pe piaţă în anul 1999. în cele ce urmează vom prezenta câteva dintre cele mai importante funcţii analitice, care extind nucleul SQL către probleme legate destul de strâns de ceea ce îndeobște este cunoscut sub titulatura
depozite de date (Data Warehouses).

7.1. Subtotaluri. ROLLUP şi GROUPING

Problema subtotalurilor în SQL a fost abordată încă din paragraful 4.6. Am convenit că, prin mai multe fraze SELECT conectate prin UNION, şi prin grupare, se poate obţine o formă rezonabilă de raport. Vom începe cu o problemă asemănătoare, căreia îi vom răspunde mai întâi printr-o interogare „clasică", astfel încât noul operator ROLLUP să aibă parte de o intrare de efect.

Care este totalul vânzărilor, pe clienţi, localităţi, judeţe şi totalul general?

Astfel de analize sunt extrem de frecvente, pentru orice gen de firmă. în lipsa unui operator de genul ROLLUP, soluţiile sunt laborioase. Să începem cu următoarea interogare DB2:

```
SELECT Judeţ, Loc, DenCl,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER
            JOIN FACTURI F  ON LF.NrFact=F.NrFact
   INNER    JOIN CLIENŢI  C  ON F.CodCl=C.CodCI
   INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost INNER
JOIN JUDEŢE J ON L.Jud=J.Jud GROUP BY Judeţ, Loc, DenCl
UNION
SELECT Judeţ, Loc,       ' Subtotal       LOCALITATE' AS       DenCl,
   SUM(Cantitate *       PretUnit *       (1+ProcTVA)) AS       Vinzari
FROM PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER
            JOIN FACTURI F  ON LF.NrFact=F.NrFact
   INNER    JOIN CLIENŢI  C  ON F.CodCl=C.CodCI
   INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost INNER
JOIN JUDEŢE J ON L.Jud=J.Jud GROUP BY Judeţ, Loc
UNION
SELECT Judeţ, ' Subtotal JUDEŢ' AS Loc, '               ' AS DenCl,
   SUM(Cantitate *       PretUnit *       (1+ProcTVA)) AS       Vinzari
FROM PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER
            JOIN FACTURI F  ON LF.NrFact=F.NrFact
   INNER    JOIN CLIENŢI  C  ON F.CodCl=C.CodCI
   INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost INNER
JOIN JUDEŢE J ON L.Jud=J.Jud GROUP BY Judeţ
UNION
SELECT ' TOTAL GENERAL' AS Judeţ, ' ' AS Loc, '                ' AS DenCl,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER
            JOIN FACTURI F  ON LF.NrFact=F.NrFact
   INNER    JOIN CLIENŢI  C  ON F.CodCl=C.CodCI
   INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost INNER
   JOIN JUDEŢE J ON L.Jud=J.Jud
ORDER BY Judeţ, Loc, DenCl
```

Rezultatul este cel din figura 7.1. Varianta funcţionează aproape identic în VFP, nefiind necesară prezenţa clauzei ORDER BY. Atât interogarea, cât şi ecranul de răspuns sunt specifice DB2, iar pentru trecerea în Oracle trebuie modificată sintaxa pentru joncţiune.
De obicei, în rapoarte, subtotalurile şi totalul general sunt afişate *după* fiecare grup, respectiv pe ultima linie a rezultatului, în timp ce interogarea precedentă plasează subtotalul la început. Plasarea normală (la sfârşitul grupului) necesită utilizarea unui caracter cu un cod ASCII suficient de mare pentru ca linia respectivă să fie ultima din grup la ordonare, după cum am văzut şi în paragraful *4,6.*

JUDEŢ	LOC	DENCL	VINZARI
1 TOTAL GENERAL			35839230.00
Iasi	Subtotal JUDEŢ		19088790.00
Iasi	Iasi	Subtotal LOCALITATE	13650490.00
Iasi	Iasi	Clienţi SRL	12490240.00
Iasi	Iasi	Client 2 SA	1160250.00
Iasi	Paşcani	Subtotal LOCALITATE	5438300.00
Iasi	Paşcani	Client 4	5438300.00
Neamţ	Subtotal JUDEŢ		6786570.00
;Neamt	Roman	Subtotal LOCALITATE	6786570.00
Neamţ	Roman	Cliente SA	6786570.00
Timiş	Subtotal JUDEŢ		2720935.00
Timiş	Timişoara	Subtotal LOCALITATE	2720935.00
Timiş	Timişoara	Client 5 SRL	1337560.00
Timiş	Timişoara	Client 7 SRL	1383375.00
Vaslui	Subtotal JUDEŢ		7242935.00
Vaslui	Vaslui	Subtotal LOCALITATE	7242935.00
Vaslui	Vaslui	Client 3 SRL	7242935.00

Figura 7.1. Subtotaluri la începutul grupurilor

Pentru a mai schimba un pic aerul, soluţia următoare este scrisă în Oracle 8i2, raportul obţinut fiind cel din figura 7.2.

```
SELECT Judeţ, Loc, DenCl,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE
P, LINIIFACT LF, FACTURI F,
    CLIENŢI C, LOCALITATI L, JUDEŢE J WHERE P.CodPr=LF.Codpr
AND LF.NrFact=F.NrFact AND
    F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP BY
Judeţ, Loc, DenCl
UNION
SELECT Judeţ, Loc, CONCAT (CHR(255),                    ' Subtotal
LOCALITATE')
AS DenCl,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE
P, LINIIFACT LF, FACTURI F,
    CLIENŢI C, LOCALITATI L, JUDEŢE J WHERE P.CodPr=LF.Codpr
AND LF.NrFact=F.NrFact AND
    F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP BY
Judeţ, Loc
UNION
SELECT Judeţ, CONCAT (CHR(255),    '    Subtotal  JUDEŢ') AS Loc,
       '  ' AS DenCl,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE
P, LINIIFACT LF, FACTURI F,
CLIENŢI C, LOCALITATI L, JUDEŢE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
    F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP BY
Judeţ
UNION
    SELECT CONCAT (CHR(255),          ' TOTAL GENERAL') AS Judeţ, '              '
        AS Loc, ' ' AS DenCl,
        SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P,
    LINIIFACT LF, FACTURI F,
        CLIENŢI C, LOCALITATI L, JUDEŢE J WHERE P.CodPr=LF.Codpr AND
    LF.NrFact=F.NrFact AND
        F.CodCl=C.CodCI AND C.CodPost=L.CodPost AND L.Jud=J.Jud
    ORDER BY Judeţ, Loc, DenCl
```

JUDEŢ	LOC	DENCL	VINZARI
Iasi	Iasi	Client 1 SRL	12490240
Iasi	Iasi	Client 2 SA	1160250
Iasi	Iasi	Subtotal LOCALITATE	13650490
Iasi	Paşcani	Client 4	5438300
Iasi	Paşcani	Subtotal LOCALITATE	5438300
Iasi	Subtotal JUDEŢ		19088790
Neant	Roman	Client 8 SA	8788570
Neamţ	Roman	Subtotal LOCALITATE	6786570
Neamţ	Subtotal JUDEŢ		6786570
Timiş	Timişoara	Client 5 SRL	1337560
Timiş	Timişoara	Client 7 SRL	1383375
Timiş	Timişoara	Subtotal LOCALITATE	2720935
Timiş	Subtotal JUDEŢ		2720935
Vaslui	Vaslui	Client 3 SRL	7242935
Vaslui	Vaslui	Subtotal LOCALITATE	7242935
Vaslui	Subtotal JUDEŢ		7242935
TOTAL GENERAL			35839230

Figura 7.2. Plasarea corectă a subtotalurilor

Pentru acest gen de probleme, SQL-99, precum şi versiunile recente ale DB2 şi Oracle, pun la dispoziţie operatorul ROLLUP, care simplifică mult lucrurile. Să începem cu o interogare lejeră, ce foloseşte un singur atribut ca argument. Fraza SELECT de mai jos se materializează, în DB2. într-un raport de genul celui din figura 7.3.

```
SELECT DenCl,
    SUM(Cantitate * PretUnit'* (1+ProcTVA)) AS Vinzari FROM PRODUSE P
    INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
    INNER JOIN FACTURI F ON LF.NrFact=F.NrFact
    INNER JOIN CLIENŢI C ON F.CodCl=C.CodCI
    INNER JOIN LOCALITATI L ON C.CodPost=L.CodPost
    INNER JOIN JUDEŢE J ON L.Jud=J.Jud GROUP BY
ROLLUP (DenCl)
ORDER BY DenCl
```

DENCL	VINZARI
Clienţi SRL	1 2490240.00
Client 2 SA	1160250.00
Client 3 SRL	7242935.00
Client 4	5438300.00
Client 5 SRL	1337560.00
Client 6 SA	6786570.00
Client 7 SRL	1383375.00
	35839230.00

Figura 7,3. Primul ROLLUP (mai simplu nu se poate)

Singurul argument va determina calculul unui total general. Pentru a răspunde punctual la problema formulată la începutul paragrafului, se includ, ca argumente ale ROLLUP, cele trei atribute, Judeţ, Loc şi DenCl.

```
SELECT Judeţ, Loc, DenCl,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P
    INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER JOIN
    FACTURI F ON LF.NrFact=F.NrFact INNER JOIN CLIENŢI C ON
    F.CodCl=C.CodCl INNER JOIN LOCALITATI L ON
    C.CodPost=L.CodPost INNER JOIN JUDEŢE J ON L.Jud=J.Jud
    GROUP BY ROLLUP (Judeţ, Loc, DenCl)
ORDER BY Judeţ, Loc, DenCl
```

în DB2 v7 se obţine situaţia din figura 7.4. Pentru binele soluţiei, este necesară şi clauza ORDER BY. Primul atribut-argument, Judeţ, determină calculul totalului general *(grand total*, cum i se spune în popor). Al doilea argument, Loc, atrage după sine însumarea vânzărilor pentru fiecare valoare distinctă a Judeţ, iar al treilea, DenCl, are ca efect calcularea a câte unui subtotal pentru fiecare combinaţie (Judeţ, Loc). Pentru ca subtotalurile şi totalul general să fie marcate explicit, se poate folosi una dintre funcţiile: VALUE, COALESCE, NVL, aceasta deoarece în rândurile respective, valorile atributelor de grupare sunt NULL. Iată soluţia DB2 care condiţionează obţinerea rezultatului corect de utilizarea clauzei ORDER BY după GROUP BY:

Figura 7.4. Subtotaluri multiple prin **ROLLUP**

JUDEŢ	LOC	DENCL	VINZARI.
Iasi	Iasi	Client 1 SRL	1 2490240.00
Iasi	Iasi	Client 2 SA	1160250.00
Iasi	Iasi		13650490.00
Iasi	Paşcani	Client 4	5438300.00
Iasi	Paşcani		5438300.00
Iasi			19088790.00
Neamţ	Roman	Client 6 SA	6786570.00
Neamţ	Roman		6786570.00
Neamţ			6786570.00
Timiş	Timişoara	Client 5 SRL	1337560.00
Timiş	Timişoara	Client 7 SRL	1383375.00
Timiş	Timişoara		2720935.00
Timiş			2720935.00
Vaslui	Vaslui	Client 3 SRL	7242935.00
Vaslui	Vaslui		7242935.00
Vaslui			7242935.00
			35839230.00

```
SELECT COALESCE(Judeţ, CHR(255)      ||   ' TOTAL   GENERAL')
      AS Judeţ,
   COALESCE(Loc, CHR(255) II ' Subtotal JUDEŢ '          ||
271    Judeţ) AS Loc,                    SQL
   COALESCE(DenCl, CHR(255)    ||   '   Subtotal    LOCALITATE '   ||
      Loc) AS DenCl,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE          P
   INNER    JOIN LINIIFACT  LF ON P.CodPr=LF.Codpr
   INNER    JOIN FACTURI F ON LF.NrFact=F.NrFact
   INNER    JOIN CLIENŢI C  ON F.CodCl=C.CodCI
   INNER    JOIN LOCALITATI L ON C.CodPost=L.CodPost
 INNER JOIN JUDEŢE J ON L.Jud=J.Jud GROUP BY
                    ROLLUP (Judeţ, Loc, DenCl)
ORDER BY Judeţ, Loc, DenCl
```

şi rezultatul în figura 7.5.

Interesant este că aceeaşi soluţie, transcrisă în sintaxa Oracle 8i2, conduce la un rezultat diferit în ceea ce priveşte liniile subtotalurilor, după cum se observă în figura 7.6.

```
SELECT NVL(Judeţ, CHR(255)      ||   ' TOTAL GENERAL') AS Judeţ,
   NVL(Loc, CHR(255)    ||   ' Subtotal JUDEŢ '        || Judeţ)
      AS Loc,
   NVL(DenCl, CHR(255)     ||   '    Subtotal LOCALITATE ' ||
      Loc) AS DenCl,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE P, LINIIFACT LF, FACTURI F, CLIENŢI C, LOCALITATI L,
JUDEŢE J WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
   F.CodCl=C.CodCI AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP BY
ROLLUP (Judeţ, Loc, DenCl)
ORDER BY Judeţ, Loc, DenCl
```

JUDEŢ '.i	, LOC	DENCL ,	VINZARI
lasi	lasi	Clienţi SRL	12490240.00
lasi	lasi	Client 2 SA	1 160250.00
lasi	lasi	y Subtotal LOCALITATE lasi	13650490.00
lasi	Paşcani	Client 4	5438300.00
lasi	Paşcani	y Subtotal LOCALITATE Paşcani	5438300.00
lasi	y Subtotal JUDEŢ lasi		19088790.00
Neamţ	Roman	Client 6 SA	6786570.00
Neamţ	Roman	Ifsubtotai LOCALITATE Roman	6786570.00
Neamţ	y Subtotal JUDEŢ Neamţ		6786570.00
Timiş	Timişoara	Client 5 SRL	1337560.00
Timiş	Timişoara	Client 7 SRL	1383375.00
Timiş	Timişoara	y Subtotal LOCALITATE Timişoara	2720935.00
Timiş	¥ Subtotal JUDEŢ Timiş		2720935.00
Vaslui	Vaslui	Client 3 SRL	7242935.00
Vaslui	Vaslui	y Subtotal LOCALITATE Vaslui	7242935.00
Vaslui	y Subtotal JUDEŢ Vaslui		7242935 00
y TOTAL GENERAL			35839230.00

Figura 7.5. ROLLUP cu marcarea explicită a subtotalurilor

JUDEŢ	LOC	DENCL	VINZARI
Iasi	Iasi	Client 1 SRL	12490240
Iasi	Iasi	Client 2 SA	1160250
Iasi	Iasi	Subtotal LOCALITATE Iasi	13650490
Iasi	Paşcani	Client 4	5438300
Iasi	Paşcani	Subtotal LOCALITATE Paşcani	5438300
Iasi	Subtotal JUDEŢ Iasi	Subtotal LOCALITATE	19088790
Neamţ	Roman	Client 6 SA	6786570
Neamţ	Roman	Subtotal LOCALITATE Roman	6786570
Neant	Subtotal JUDEŢ Neamţ	Subtotal LOCALITATE	6786570
Timiş	Timişoara	Client 5 SRL	1337560
Timiş	Timişoara	Client 7 SRL	1383375
Timiş	Timişoara	Subtotal LOCALITATE Timişoara	2720935
Timiş	Subtotal JUDEŢ Timiş	Subtotal LOCALITATE	2720935
Vaslui	: Vaslui	Client 3 SRL	7242935
Vaslui	Vaslui	Subtotal LOCALITATE Vaslui	7242935
Vaslui	Subtotal JUDEŢ Vaslui	Subtotal LOCALITATE	7242935
TOTAL GENERAL	Subtotal JUDEŢ	Subtotal LOCALITATE	35839230

Figura 7.6, Rezultat diferit al aplicării ROLLUP în Oracle 8i2, faţă de DB2 v7

Raportul din figura 7.6 obţinut în Oracle are un inconvenient: la subtotaluri pe judeţe apare, oarecum anapoda, şi un *Subtotal LOCALITATE*. Lucrul acesta poate fi corectat printr-o structură de tip CASE, deşi nici DECODE-ul nu-i de lepădat.

```
SELECT NVL(Judeţ, CHR(255)      ||  '  TOTAL GENERAL') AS Judeţ,
   CASE WHEN Judeţ IS NULL
      THEN '   !                                    •
      ELSE NVL(Loc, CHR(255)    ||  '     Subtotal  JUDEŢ '  ||
         Judeţ)
   END AS Loc,
   CASE WHEN Loc IS NULL THEN '
            '
      ELSE NVL(DenCl, CHR(255)     ||  ' Subtotal LOCALITATE '
         II Loc)
   END AS DenCl,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE P, LINIIFACT LF, FACTURI F, CLIENŢI C, LOCALITATI L,
JUDEŢE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
   F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP BY
ROLLUP (Judeţ, Loc, DenCl)
ORDER BY Judeţ, Loc, DenCl
```

De.această dată, lista are chiar forma (şi conţinutul) ca în figura 7.5.

Combinarea opţiunilor ROLLUP şi GROUPING

Structura alternativă de tip CASE poate utiliza şi o nouă funcţie introdusă în SQL-99 - GROUPING. Argumentul acesteia este coloana de grupare, rezultatul întors fiind 1 atunci când coloana respectivă este inclusă într-un grup de agregare superior, sau 0 pentru liniile „normale" (din afara subtotalurilor). Pentru edificare, vezi SELECT-ul următor, ce produce rezultatul din figura 7.7.

```
SELECT Judeţ, Loc, DenCl,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari,
   GROUPING (Judeţ) AS Grup_Judet,
   GROUPING (Loc) AS Grup_Loc,
   GROUPING (DenCl) AS Grup_DenCl FROM PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER JOIN
   FACTURI F ON LF.NrFact=F.NrFact INNER JOIN CLIENŢI C ON
   F.CodCl=C.CodCI INNER JOIN LOCALITATI L ON
   C.CodPost=L.CodPost INNER JOIN JUDEŢE J ON L.Jud=J.Jud GROUP
   BY ROLLUP (Judeţ, Loc, DenCl)
ORDER BY Judeţ, Loc, DenCl
```

JUDEŢ	LOC	DENCL	VINZARI	GRUP_JUDET	GRUP LOC	GRUP_DENCL
Iasi	Iasi	Clienţi SRL	12490240.OCr	0	0	0
Iasi	Iasi	Client 2 SA	1160250.00	0	0	0
Iasi	Iasi		13650490.00	0	0	1
Iasi	Paşcani	Client 4	5438300.00	0	0	0
Iasi	Paşcani		5438300 00	0	0	1
Iasi			19088790.00	0	1	1
Neamţ	Roman	Client 6 SA	6786570.00	0	0	0
Neamţ	Roman		6786570.00	0	0	1
Neamţ			5786570.00	0	1	1
Timiş	Timişoara	Client 5 SRL	1337560.00	0	0	0
Timiş	Timişoara	Client 7 SRL	1383375.00	0	0	0
Timiş	Timişoara		2720935.00	0	0	1
Timiş			2720935.00	0	1	1
Vaslui	Vaslui	Client 3 SRL	7242935.00	0	0	0
Vaslui	Vaslui		7242935.00	0	0	1
Vaslui			7242935.00	0	1	1
			35839230.00	1	1	1

Figura 7.7. Valori întoarse de funcţia GROUPING

Valoarea 1 întoarsă de funcţia GROUPING pentru un atribut indică un subtotal pentru acel atribut, relativ la atributele din stânga (în ordinea declarată în clauza ROLLUP). Pe baza funcţiei GROUPING, rezultatul din figura 7.5 poate fi obţinut în DB2 şi astfel:

```
SELECT
   CASE
      WHEN GROUPING (Judeţ) = 1 THEN CHR(255)        ||
         ' TOTAL GENERAL'
      ELSE Judeţ END
            AS Judeţ,
   CASE
      WHEN GROUPING (Loc) = 1 THEN CHR(255)       ||
         ' Subtotal JUDEŢ '|| Judeţ ELSE Loc
   END AS Loc,
   CASE
      WHEN GROUPING (DenCl) = 1 THEN CHR(255)
         I I ' Subtotal LOCALITATE '          |          |          DenCl
     ELSE DenCl END
          AS DenCl,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER JOIN
   FACTURI F ON LF.NrFact=F.NrFact INNER JOIN CLIENŢI C ON
   F.CodCl=C.CodCl INNER JOIN LOCALITATI L ON
   C.CodPost=L.CodPost INNER JOIN JUDEŢE J ON L.Jud=J.Jud
   GROUP BY ROLLUP (Judeţ, Loc, DenCl)
ORDER BY Judeţ, Loc, DenCl
```

ROLLUP-uri parţiale

Este de dorit, uneori, ca subtotalizarea atributelor de grupare să fie parţială: fie nu interesează totalul general, fie subtotalizarea este necesară numai pentru anumite atribute sau grupuri de atribute. Complicăm un pic problema, în sensul că acum vânzările interesează pe facturi, clienţi, localităţi şi judeţe. Subtotalurile însă, trebuie calculate numai la nivel de client şi localitate. Interogarea ce furnizează răspunsul (figura 7.8) este în Oracle 8i2:

JUDEŢ	LOC	DENCL	FACTURA		VINZARI
Iasi	Iasi	Client 1 SRL	1111		5399625
Iasi	Iasi	Client 1 SRL	1115		1651125
Iasi	Iasi	Client 1 SRL	1117		2320500
Iasi	Iasi	Client 1 SRL	1118		2052750
Iasi	Iasi	Client 1 SRL	1120		1066240
Iasi	Iasi	Client 1 SRL	Subtotal CLIENT	Ciient 1 SRL	12490240
Iasi	Iasi	Client 2 SA	1113		1160250
Iasi	Iasi .	Client 2 SA	Subtotal CLIENT	Client 2 SA	1160250
Iasi	Iasi	Subtotal LOCALITATE	Iasi		13650490
Iasi	Paşcani	Client 4	1121		5438300
iasi	Paşcani	Client 4	Subtotal CLIENT	Client 4	5438300
Iasi	Paşcani	Subtotal LOCALITATE	Paşcani		5438300
Neamţ	Roman	Client 6 SA	1114		6786570
Neamţ	Roman	Client 6 SA	Subtotal CLIENT	Client 6 SA	6786570
Neamţ		Subtotal LOCALITATE	Roman		6786570
Timiş	Timişoara	Client 5 SRL	1112		1337560
Timiş	Timişoara	Client 5 SRL	Subtotal CLiENT	Client S SRL	1337560
Timiş	Timişoara	Client 7 SRL	1116		1383375
Timiş	Timişoara	Client 7 SRL	Subtotal CLIENT	Client 7 SRL	1383375
Timiş	Timişoara	Subtotal LOCALITATE	Timişoara		2720935
Vaslui	Vaslui	Client 3 SRL	1119		7242935
Vaslui	Vaslui	Client 3 SRL	Subtotal CLIENT	Client 3 SRL	7242935
Vaslui	Vaslui	Subtotal LOCALITATE	Vaslui		7242935

Figura 7.8. ROLLUP parţial

```
SELECT NVL(Judeţ, CHR(255) II ' TOTAL GENERAL') AS Judeţ, CASE WHEN
    Judeţ IS NULL THEN '     '
    ELSE NVL(Loc, CHR(255) II ' Subtotal JUDEŢ ' I i Judeţ)
    END AS Loc,
    CASE WHEN Loc IS NULL THEN ''
    ELSE NVL(DenCl, CHR(255) il ' Subtotal LOCALITATE '
        II Loc)
    END AS DenCl,
    CASE WHEN DenCl IS NULL THEN '       '
    ELSE NVL(TO_CHAR(F.NrFact,'99999999'), CHR(255)          ||
        ' Subtotal CLIENT '          ||  RTRIM(DenCl))
    END AS Factura,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENŢI C,
LOCALITATI L, JUDEŢE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
    F.CodCl=C.CodCI AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP BY Judeţ,
Loc, ROLLUP (DenCl, F.NrFact)
ORDER BY Judeţ, Loc, DenCl, F.NrFact
```

Prin clauza GROUP BY Judeţ, Loc, ROLLUP (DenCl, F.NrFact) se vor calcula două tipuri de subtotaluri:

- unul pentru valori distincte ale combinaţiei (Judeţ, Loc şi DenCl) şi
- altul pentru valori distincte ale combinaţiei (Judeţ, Loc).

Reţinem deci: o coloană declarată în ROLLUP determină calcularea subtotalurilor pentru valori distincte ale celorlalte coloane. Astfel, interogarea următoare produce în Oracle 8i2 lista din figura 7.9.

```
SELECT NVL(Judeţ, CHR(255)          ||  ' TOTAL GENERAL') AS Judeţ,
    CASE WHEN Judeţ IS NULL THEN '
        '
        ELSE NVL(Loc, CHR(255)          ||  ' Subtotal JUDEŢ '          ||
            Judeţ)
        END AS Loc,
    CASE WHEN Loc IS NULL
```

```
    THEN '      '
    ELSE NVL(DenCl, CHR(255)              ||    '         Subtotal    LOCALITATE   '
        II    Loc)
  END AS DenCl,
  CASE WHEN DenCl IS NULL THEN
    '              '
    ELSE NVL(TO_CHAR(F.NrFact,'99999999'), CHR(255)                    ||
      ' Subtotal CLIENT '                ||    RTRIM(DenCl))
  END AS Factura,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P,
                                    LINII_FACTURI LF, FACTURI F, CLIENŢI C,
LOCALITATI L, JUDEŢE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
    F.CodCl=C.CodCI AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP BY Judeţ,
ROLLUP(Loc, DenCl), F.NrFact
```


Judeţ	Loc	DenCl	Factura	Vinzari
Iasi	Iasi	Client 1 SRL	1111	5399625
Iasi	Iasi	Subtotal LOCALITATE Iasi		5399625
Iasi	Subtotal JUDEŢ Iasi			5399625
iasi	Iasi	Client 2 SA	1113	1160250
Iasi	Iasi	Subtotal LOCALITATE Iasi		1160250
Iasi	Subtotal JUDEŢ Iasi			1160250
Iasi	Iasi	Client 1 SRL	1115	1651125
Iasi	Iasi	Subtotal LOCALITATE Iasi		1651125
Iasi	Subtotal JUDEŢ Iasi			1651125
Iasi	Iasi	Client 1 SRL	1117	2320500
Iasi	Iasi	Subtotal LOCALITATE Iasi		2320500
Iasi	Subtotal JUDEŢ Iasi			2320500
Iasi	Iasi	Client 1 SRL	1118	2052750
Iasi	Iasi	Subtotal LOCALITATE Iasi		2052750
Iasi	Subtotal JUDEŢ Iasi			2052750
Iasi	Iasi	Client 1 SRL	1120	1066240
Iasi	Iasi	Subtotal LOCALITATE Iasi		1066240
Iasi	Subtotal JUDEŢ Iasi			1066240
Iasi	Paşcani	Client 4	1121	5438300
Iasi	Paşcani	Subtotal LOCALITATE Paşcani		5438300
Iasi	Subtotal JUDEŢ Iasi			5438300
Neamţ	Roman	Client 6 SA	1114	6786570
Neamţ	Roman	Subtotal LOCALITATE Roman		6786570
Neamţ	Subtotal JUDEŢ Neamţ			6786570
Timiş	Timişoara	Client 5 SRL	1112	1337560
Timiş	Timişoara	Subtotal LOCALITATE Timişoara		1337560
Timiş	Subtotal JUDEŢ Timiş			1337560
Timiş	Timişoara	Client 7 SRL	1116	1383375
Timiş	Timişoara	Subtotal LOCALITATE Timişoara		1383375
Timiş	Subtotal JUDEŢ Timiş			1383375
Vaslui	Vaslui	Client 3 SRL	1119	7242935
Vaslui	Vaslui	Subtotal LOCALITATE Vaslui		7242935
Vaslui	Subtotal JUDEŢ Vaslui			7242935

Figura 7.9. Un alt ROLLUP parţial, dar ceva mai curios

Rezultatul este destul de anapoda, dar nici interogarea nu-i mai prejos! Practic, după fiecare valoare distinctă a combinaţiei atributelor din GROUP BY se calculează un subtotal pe judeţ-localitate şi altul pe judeţ (deoarece atributele din ROLLUP sunt DenCl şi Loc).

Varianta DB2, obţinută prin schimbarea funcţiei NVL cu VALUE, conduce la aceleaşi linii-rezultat, dar într-o cu totul altă ordine, după cum se observă în figura 7.10.

```
SELECT VALUE(Judeţ, CHR(255)              ||     ' TOTAL GENERAL') AS Judeţ,
  CASE WHEN Judeţ IS NULL THEN
    '              '
    ELSE VALUE(Loc, CHR(255)              ||  ' Subtotal JUDEŢ '
      I| Judeţ)
  END AS Loc,
  CASE WHEN Loc IS NULL THEN '
                '
    ELSE VALUE(DenCl, CHR(255) II
      ' Subtotal LOCALITATE '                 || Loc)
  END AS DenCl,
    CASE WHEN DenCl IS NULL THEN '      '
```

```
        ELSE VALUE(CAST (F.NrFact AS CHAR(8)),          CHR(255)    ||    '
Subtotal CLIENT '        ||  RTRIM(DenCl))
    END AS Factura,
        SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
            PRODUSE P, LINIIFACT LF, FACTURI F, CLIENŢI C,
LOCALITATI L, JUDEŢE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP
BY Judeţ, ROLLUP(Loc, DenCl), F.NrFact
```


JUDEŢ	LOC	DENCL	FACTURA	VINZARI
lasi	Subtotal JUDEŢ lasi			5399625
lasi	Subtotal JIJDET lasi			1160250
lasi	Subtotal JUDEŢ lasi			1651125
lasi	Subtotal JUDEŢ lasi			2320500
lasi	Subtotal JUDEŢ lasi			2052750
lasi	Subtotal JUDEŢ lasi			1066240
lasi	Subtotal JUDEŢ lasi			5438300
lasi	lasi	Subtotal LOCALITATE lasi		5399625
lasi	lasi	Subtotal LOCALITATE lasi		1160250
lasi	lasi	Subtotal LOCALITATE lasi		1651125
lasi	lasi	Subtotal LOCALITATE lasi		2320500
lasi	lasi	Subtotal LOCALITATE lasi		2052750
lasi	lasi	Subtotal LOCALITATE lasi		1066240
lasi	lasi	Client 1 SRL	1111	5399625
lasi	lasi	Client 1 SRL	1115	1651125
lasi	lasi	Client 1 SRL	1117	2320500
lasi	lasi	Client 1 SRL	1118	2052750
lasi	lasi	Client 1 SRL	1120	1066240
lasi	lasi	Client 2 SA	1113	1160250
lasi	Paşcani	Subtotal LOCALITATE Paşcani		5438300
lasi	Paşcani	Client 4	1121	5438300
Neamţ	Subtotal JUDEŢ Neamţ			6786570
Neamţ	Roman	Subtotal LOCALITATE Roman		6786570
Neamţ	Roman	Client 6 SA	1114	6786570
Timiş	Subtotal JUDEŢ Timiş			1337560
Timiş	Subtotal JUDEŢ Timiş			1383375
Timiş	Timişoara	Subtotal LOCALITATE Timişoara		1337560
Timiş	Timişoara	Subtotal LOCALITATE Timişoara		1383375
Timiş	Timişoara	Client 5 SRL	1112	1337560
Timiş	Timişoara	Client 7 SRL	1116	1383375
Vaslui	Subtotal JUDEŢ Vaslui			7242935
Vaslui	v'aslui	Subtotal LOCALITATE Vaslui		7242935
Vaslui IVaslui		Client 3 SRL	1119	7242935

Figura 7.10. Dispunerea liniilor în DB2 la un ROLLUP parţial

Practic, este cu neputinţă să obţinem aceeaşi dispunere a liniilor ca în Oracle. In finalul paragrafului, pentru o deplină edificare asupra mecanismului de execuţie a operaţiunii de ROLLUP parţial, prezentăm o ultimă interogare Oracle 8i2 şi rezultat •' acesteia (figura 7.11).

JUDEŢ	LOC	DENCL	FACTURA	VINZARI
Iasi	las	Client 1 SRL	1111	5399625
Iasi	las	Client 1 SRL	1115	1651125
Iasi	las	Client 1 SRL	1117	2320500
Iasi	las	Client 1 SRL	1118	2052750
Iasi	las	Clienţi SRL	1120	1066240
Iasi	las	Client 1 SRL	Subtotal CLIENT Client 1 SRL	12490240
Iasi	Subtotal JUDEŢ Iasi		Subtotal CLIENT Client 1 SRL	12490240
Iasi	Iasi	Client 2 SA	1113	1160250
Iasi	las	Client 2 SA	Subtotal CLIENT Client 2 SA	1160250
Iasi	Subtotal JUDEŢ Iasi		Subtotal CLIENT Client 2 SA	1160250
Iasi	Paşcani	Client 4	1121	5438300
Iasi	Paşcani	Client 4	Subtotal CLIENT Client 4	5438300
Iasi	Subtotal JUDEŢ Iasi		Subtotal CLIENT Client 4	5438300
Neamţ	Roman	Client 6 SA	1114	6786570
Neamţ	Roman	Client 6 SA	Subtotal CLIENT Client 6 SA	6786570
Neamţ	Subtotal JUDEŢ Neamţ		Subtotal CLIENT Client 6 SA	6788570
Timiş	Timişoara	Client 5 SRL	1112	1337560
Timiş	Timişoara	Client 5 SRL	Subtotal CLIENT Client S SRL	1337560
Timiş	Subtotal JUDEŢ Timiş		Subtotal CLIENT Client 5 SRL	1337560
Timiş	Timişoara	Client 7 SRL	1116	1383375
Timiş	Timişoara	Client 7 SRL	Subtotal CLIENT Client 7 SRL	1383375
Timiş	Subtotal JUDEŢ Timiş		Subtotal CLIENT Client 7 SRL	1383375
Vaslui	Va slui	Client 3 SRL	1119	7242935
Vaslui	Vaslui	Client 3 SRL	Subtotal CLIENT Client 3 SRL	7242935
Vaslui	Subtotal JUDEŢ Vaslui		Subtotal CLIENT Client 3 SRL	7242935

Figura 7.11. Un ultim exemplu de ROLLUP parţial în Oracle 8i2

```
SELECT NVL(Judeţ, ' TOTAL GENERAL') AS Judeţ,
   CASE WHEN Judeţ IS NULL THEN ' '
      ELSE NVL(Loc, ' Subtotal JUDEŢ '          ||    Judeţ)
   END AS Loc,
   CASE WHEN Loc IS NULL THEN '  '
      ELSE NVL(DenCl, ' Subtotal LOCALITATE '          || Loc)
   END AS DenCl,
   CASE WHEN DenCl IS NULL THEN '
            '
      ELSE NVL(TO_CHAR(F.NrFact, '99999999') , CHR(255)          ||
         ' Subtotal CLIENT '       || RTRIM(DenCl))
   END AS Factura,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P,
LINIIFACT LF, FACTURI F, CLIENŢI C, LOCALITATI L, JUDEŢE J
WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND F.CodCl=C.CodCI AND
C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP BY Judeţ, ROLLUP(Loc,
F.NrFact), DenCl
```

7.2. Analize multidimensionale. Operatorii CUBE şi GROUPING SETS

De multe ori, pentru a produce o impresie mai puternică partenerilor de discuţii, analiza datelor trebuie să fie una multidimensională. Apanaj, prin excelenţă, al... Excel-ului sau pretenţioaselor instrumente OLAP, acest paragraf este menit să demonstreze că, atât prin SQL-ul clasic (SQL-92), dar mai ales prin SQL-99, sunt create condiţii ideale pentru acest gen de operaţiuni.

Pentru început, interesează vizualizarea vânzărilor pe două axe, *produse* şi *clienţi,* ca în figura 7.12. întrucât *produse* şi *clienţi* sunt două variabile independente, vor exista patru variante de grupare a datelor:

» grupare după client şi produs,

- grupare numai după client,
- grupare numai după produs şi
- un grup pentru total general.

O soluţie în sintaxa SQL-92 este impresionantă, ca întindere, iar dacă analiza se realizează după trei, patru... variabile, impresionantul se transformă în halucinant. Iată prima interogare DB2 pentru obţinerea unui raport de genul celui din figura 7.12.

```
SELECT DenCl AS Client, DenPr AS Produs,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
```

```
PRODUSE        P
   INNER    JOIN LINIIFACT  LF ON P.CodPr=LF.Codpr
   INNER    JOIN FACTURI F ON LF.NrFact=F.NrFact
   INNER    JOIN CLIENŢI  C ON F.CodCl=C.CodCl
GROUP BY DenCl, DenPr
UNION
SELECT DenCl AS Client, CHR(255) AS Produs,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER
   JOIN           FACTURI F ON LF.NrFact=F.NrFact
   INNER JOIN    CLIENŢI  C ON F.CodCl=C.CodCl
GROUP BY DenCl
UNION
SELECT CHR(255) AS Client, DenPr AS Produs,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE P                                    •
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr                    .
   INNER JOIN    FACTURI F ON LF.NrFact=F.NrFact
   INNER JOIN    CLIENŢI  C ON F.CodCl=C.CodCl
GROUP BY DenPr
UNION
SELECT CHR(255) AS Client, CHR(255) AS Produs,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE P                              •
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact INNER
JOIN CLIENŢI C ON F.CodCl=C.CodCl ORDER BY
Client, Produs
```

CLIENT	PRODUS	VINZARI j
Clienţi SRL	Produs 1	3385550.00!
Clienţi SRL	Produs 2	5237190.00
Clienţi SRL	Produs 5	3867500.00!
Clienţi SRL		12490240,00!
Client 2 SA	Produs 2	1160250.00!
Client 2 SA		1160250.00!
Client 3 SRL	Produs 2	453985.00!
Client 3 SRL	Produs 3	333200.00!
Client 3 SRL	Produs 4	833000.00 >
Client 3 SRL	Produs 5	5622750.00!
Client 3 SRL		7242935.00!
Client 4	Produs 2	1249500.00 i
Client 4	Produs 5	4188800.00!
Client 4		5438300.00;
Client 5 SRL	Produs 2	980560.00!
Client 5 SRL	Produs 3	357000.00!
Client 5 SRL		1 337560.00 j
Client 6 SA	Produs 2	89131 0.00
Client 6 SA	Produs 4	564060.001
Client 6 SA	Produs 5	5331200.001
Client 6 SA		6786570.00!
Client 7 SRL	Produs 2	1383375.00!
Client 7 SRL		1383375.00)
	Produs 1	338555000!
	Produs 2	11356170.00!
	Produs 3	690200.00!
	Produs 4	1397060.00!
	Produs 5	19010250.00!
		35839230.00 i

Figura 7.12. Analiză pe două dimensiuni

Pentru astfel de situaţii există un operator cuceritor prin simplitate: `CUBE`. Cu ajutorul acestuia, se poate redacta următoarea variantă DB2, valabilă şi Oracle 8i2 (schimbând sintaxa j oncţiunii):

```
SELECT DenCl AS Client, DenPr AS Produs,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE P
```

```
INNER   JOIN LINIIFACT    LF ON  P.CodPr=LF.Codpr
INNER   JOIN FACTURI  F    ON LF.NrFact=F.NrFact
INNER   JOIN CLIENŢI  C    ON F.CodCl=C.CodCl
GROUP BY CUBE (DenCl, DenPr )
ORDER BY DenCl, DenPr
```

Un plus de atractivitate a rezultatului presupune folosirea clauzei GROUPING astfel:

```
SELECT
   CASE GROUPING (DenCl)
      WHEN 1 THEN CASE GROUPING (DenPr)
            WHEN 1 THEN CHR(255)    || CHR(255)       ||
               ' TOTAL '
            ELSE CHR(255)    ||   '    Total  PRODUS'
            END
      ELSE DenCl
      END AS Client,                    -
      CASE GROUPING (DenPr)
         WHEN 1 THEN CASE GROUPING (DenCl)
               WHEN 1 THEN CHR(255)    | [ CHR(255)   |  |
                  ' GENERAL '
               ELSE CHR(255)    ||    ' Total CLIENT '
               END ELSE DenPr END AS Produs,    ■
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE          P
   INNER    JOIN LINIIFACT  LF  ON P.CodPr=LF.Codpr
   INNER    JOIN FACTURI F ON LF.NrFact=F.NrFact
   INNER    JOIN CLIENŢI  C ON F.CodCl=C.CodCI
GROUP BY CUBE (DenCl, DenPr )
                              ORDER BY DenCl, DenPr
```


Drept răsplată, forma raportului este cea din figura 7.13. Ultima linie este cea a totalului general. Se cuvine de adăugat că atributele care definesc dimensiunile de analiză trebuie să fie de acelaşi tip. Varianta Oracle este în acelaşi gen, doar că, în afara joncţiunii, trebuie modificat şi modul de scriere a secvenţei CASE din fraza DB2:

```
SELECT
   CASE WHEN GROUPING (DenCl) = 1
      THEN CASE WHEN GROUPING (DenPr) = 1
         THEN CHR(255)    ||   CHR(255)   ||    ' TOTAL '
         ELSE CHR(255)    ||   ' Total PRODUS'
         END ELSE
      DenCl END AS
      Client,
   CASE WHEN GROUPING (DenPr) = 1
      THEN CASE WHEN GROUPING (DenCl) = 1
         THEN CHR(255)    || CHR(255)      ||    ' GENERAL '
         ELSE CHR(255)    ||    ' Total CLIENT '
                        END ELSE DenPr END AS Produs,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE P, LINIIFACT LF, FACTURI F, CLIENŢI C, LOCALITATI
L, JUDEŢE J WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact
AND
   F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP BY
CUBE (DenCl, DenPr )
ORDER BY DenCl, DenPr
```

CLIENT	PRODUS	VINZARI
Clienţi SRL	Produs 1	3385550.00
Clienţi SRL	Produs 2	5237190.00
Clienţi SRL	Produs 5	3867500.00
Clienţi SRL	y Total CLIENT	12490240.00
Client 2 SA	Produs 2	1160250.00
Client 2 SA	yTotal CLIENT	1 160250.00
Client 3 SRL	Produs 2	453985.00
Client 3 SRL	Produs 3	333200.00
Client 3 SRL	Produs 4	833000.00
Client 3 SRL	Produs 5	5622750.00
Client 3 SRL	yTotal CLIENT	7242935.00
Client 4	Produs 2	1249500.00
Client 4	Produs 5	4188800.00
Client 4	yTotal CLIENT	5438300.00
Client 5 SRL	Produs 2	980560.00
Client 5 SRL	Produs 3	357000.00
Client 5 SRL	yTotal CLIENT	1337560.00
Client 6 SA	Produs 2	891310.00
Cliente SA	Produs 4	564060.00
Cliente SA	Produs 5	5331200.00
Client 6 SA	yTotal CLIENT	6786570.00
Client 7 SRL	Produs 2	1383375.00
Client 7 SRL	yTotal CLIENT	1383375.00
yTotal PRODUS	Produs 1	3385550.00
y Total PRODUS	Produs 2	11356170.00
y Total PRODUS	Produs 3	690200.00
y Total PRODUS	Produs 4	1397060.00
y Total PRODUS	Produs 5	19010250.00
WTOTAL	yy GENERAL	35839230.00

Figura 7,13. Variantă DB2 de utilizare a operatorului CUBE

Atunci când analiza se întreprinde pe trei axe: *clienţi*, *produse* şi *zile*, interogarea DB2 se schimbă astfel:

```
SELECT
   CASE GROUPING (DenCl)
      WHEN 1 THEN CONCAT (CHR(255),     ' subtotal ')
    ELSE DenCl END
          AS Client,
   CASE GROUPING (DenPr)
      WHEN 1 THEN CONCAT (CHR(255), 'subtotal ')
    ELSE DenPr END
          AS Produs,
```

```
   CASE GROUPING (CAST( DataFact AS CHAR(14) ) )
      WHEN 1 THEN CONCAT (CHR(255),    '      subtotal   ')
      ELSE CAST( DataFact AS CHAR(14) )
   END AS Ziua,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE P
INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr
INNER JOIN FACTURI F ON LF.NrFact=F.NrFact INNER
JOIN CLIENŢI C ON F.CodCl=C.CodCI
GROUP BY CUBE (DenCl, DenPr, CAST( DataFact AS CHAR(14)          ) )
ORDER BY DenCl, DenPr, CAST (DataFact AS CHAR(14) )
```

Prin CAST s-a realizat conversia din dată calendaristică în şir de caractere. Rezultatul, fiind mai lung de o pagină, este afişat sub formă de tabel. Cu acest prilej, l-am mai cosmetizat un pic.

Tabelul 7.1. Analiză pe trei dimensiuni *

CLIENT	PRODUS	ZIUA	VINZARI
Client 1 SRL	Produs 1	01/08/2000	595000
Client 1 SRL	Produs 1	03/08/2000	1130500
Client 1 SRL	Produs 1	04/08/2000	1660050
Client 1 SRL	Produs 1	subtotal	3385550
Client 1 SRL	Produs 2	01/08/2000	937125
Client 1 SRL	Produs 2	02/08/2000	1651125
Client 1 SRL	Produs 2	03/08/2000	1190000
Client 1 SRL	Produs 2	04/08/2000	392700
Client 1 SRL	Produs 2	07/08/2000	1066240
Client 1 SRL	Produs 2	subtotal	5237190
Client 1 SRL	Produs 5	01/08/2000	3867500
Client 1 SRL	Produs 5	subtotal	3867500
Client 1 SRL	subtotal	01/08/2000	5399625
Client 1 SRL	subtotal ,	02/08/2000	1651125
Client 1 SRL	subtotal	03/08/2000	2320500
Client 1 SRL	subtotal	04/08/2000	2052750
Client 1 SRL	subtotal	07/08/2000	1066240
Client 1 SRL	subtotal	subtotal	12490240
Client 2 SA	Produs 2	01/08/2000	1160250
Client 2 SA	Produs 2	subtotal	1160250
Client 2 SA	subtotal	01/08/2000	1160250
Client 2 SA	subtotal	subtotal	1160250
Client 3 SRL	Produs 2	07/08/2000	453985
Client 3 SRL	Produs 2	subtotal	453985
Client 3 SRL	Produs 3	07/08/2000	333200
Client 3 SRL	Produs 3	subtotal	333200
Client 3 SRL	Produs 4	07/08/2000	833000

subtotal	**Produs 4**	**01/08/2000**	**564060**
subtotal	**Produs 4**	**07/08/2000**	**833000**
subtotal	**Produs 4**	**subtotal**	**1397060**
subtotal	**Produs 5**	**01/08/2000**	**9198700**
subtotal	**Produs 5**	**07/08/2000**	**9811550**
subtotal	**Produs 5**	**subtotal**	**19010250**
subtotal	**subtotal**	**01/08/2000**	**14684005**
subtotal	**subtotal**	**02/08/2000**	**3034500**
subtotal	**subtotal**	**03/08/2000**	**2320500**
subtotal	**subtotal**	**04/08/2000**	**2052750**
subtotal	**subtotal**	**07/08/2000**	**13747475**
subtotal	**subtotal**	**subtotal**	**35839230**

Obţinerea aceluiaşi rezultat utilizând operatorul CUBE presupune, în Oracle 8i2, următoarea interogare:

```
SELECT
    CASE WHEN GROUPING (DenCl) = 1 THEN ' subtotal '
       ELSE DenCl END
             AS Client,
    CASE WHEN GROUPING (DenPr) = 1 THEN ' subtotal '
       ELSE DenPr END
             AS Produs,
    CASE WHEN GROUPING (TO_CHAR(DataFact,'DD-MM-YYYY')) = 1 THEN ' subtotal '
       ELSE TO_CHAR(DataFact,'DD-MM-YYYY')
    END AS Ziua,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P,
LINIIFACT LF, FACTURI F, CLIENŢI C,
    LOCALITATI L, JUDEŢE J WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
    F.CodCl=C.CodCI AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP BY CUBE (DenCl,
DenPr, TO_CHAR(DataFact,'DD-MM-YYYY'))
```

In absenţa operatorului CUBE, SQL-92 cere ceva efort de scriere, deşi logica interogării este relativ simplă:

```
SELECT DenCl AS Client, DenPr AS Produs,
    CAST( DataFact AS CHAR(14)              ) AS Ziua,                    '
    SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P
    INNER      JOIN LINIIFACT   LF ON P.CodPr=LF.Codpr
    INNER      JOIN FACTURI   F ON LF.NrFact=F.NrFact
    INNER      JOIN CLIENŢI   C ON F.CodCl=C.CodCI
GROUP BY DenCl, DenPr, CAST( DataFact AS CHAR(14) )
UNION
SELECT DenCl AS Client, DenPr AS Produs,
   CHR(255)   ||  '  subtotal ' AS Ziua,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE
                  P
   INNER JOIN   LINIIFACT  LF ON P.CodPr=LF.Codpr
   INNER JOIN   FACTURIF  ON LF.NrFact=F.NrFact
   INNER JOIN   CLIENŢI C  ON F.CodCl=C.CodCI
GROUP BY DenCl, DenPr UNION
SELECT DenCl AS Client, CHR(255)       ||  '    subtotal  ' AS  Produs,
   CAST( DataFact AS CHAR(14)      ) AS Ziua,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER
   JOIN            FACTURI    F ON   LF.NrFact=F.NrFact
   INNER JOIN      CLIENŢI    C ON   F.CodCi=C.CodCI
GROUP BY DenCl, CAST( DataFact AS CHAR(14)       )
UNION
SELECT CHR(255)    | i ' subtotal ' AS Client, DenPr AS Produs,
   CAST( DataFact AS CHAR(14) ) AS Ziua,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER
   JOIN            FACTURI    F ON   LF.NrFact=F.NrFact
   INNER JOIN      CLIENŢI    C ON   F.CodCl=C.CodCI
```

```
GROUP BY Den-Pr, CAST ( DataFact AS CHAR (14)      )
UNION
SELECT DenCi AS    Client, CHR(255) i |      'subtotal       ' AS  Produs,
   CHR(255)   ||  '   subtotal ' AS     Ziua,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER
   JOIN          FACTURI  F    ON  LF.NrFact=F.NrFact
   INNER JOIN    CLIENŢI  C    ON  F.CodCl=C.CodCI
GROUP BY DenCl UNION
SELECT CHR(255)   ||    1 subtotal   ' AS Client,     DenPr   AS  Produs,
   CHR(255)   ||  '   subtotal ' AS     Ziua,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER
   JOIN          FACTURI  F    ON  LF.NrFact=F.NrFact
   INNER JOIN    CLIENŢI  C    ON  F.CodCl=C.CodCI
GROUP BY DenPr UNION
SELECT CHR(255)   !|  '  subtotal ' AS Client, CHR(255)          ||
       ' subtotal ' AS Produs, CAST( DataFact AS CHAR(14) )
       AS Ziua,
```

```
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER
   JOIN          FACTURI F ON LF.NrFact=F.NrFact
   INNER JOIN    CLIENŢI C ON F.CodCl=C.CodCl
GROUP BY CAST( DataFact AS CHAR(14) )
UNION
SELECT CHR(255)    ||  '    subtotal  ' AS Client, CHR(255)       ||
   ' subtotal ' AS Produs, CHR(255)            ||  '    subtotal    '
      AS Ziua,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER
   JOIN          FACTURI F ON LF.NrFact=F.NrFact
   INNER JOIN    CLIENŢI C ON F.CodCl=C.CodCl
ORDER BY Client, Produs, Ziua
```


Operatorul GROUPING SETS

Uneori, prea multă detaliere, precum cea din tabelul de mai sus, devine obositoare. Prin GROUPING SETS se poate regla granularitatea agregărilor. Dacă, spre exemplu, în interogarea anterioară se doreşte urmărirea zilnică a vânzărilor pentru fiecare client (şi toate produsele) şi pentru fiecare produs (şi toţi clienţii), se poate recurge la varianta următoare:

```
SELECT
   CASE GROUPING (DenCl)
      WHEN 1 THEN CONCAT (CHR(255),    '-subtotal-       ')
      ELSE DenCl END AS Client,
   CASE GROUPING (DenPr)
      WHEN 1 THEN CONCAT (CHR(255),    '-subtotal-       ')
      ELSE DenPr END AS Produs,
   CASE GROUPING (CAST( DataFact AS CHAR(14) )      )
      WHEN 1 THEN CONCAT (CHR(255),    '-subtotal-       ')
      ELSE   CAST( DataFact  AS CHAR(14)   ) END AS Ziua,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
PRODUSE P
   INNER   JOIN LINIIFACT LF     ON P.CodPr=LF.Codpr
   INNER   JOIN FACTURI F ON       LF.NrFact=F.NrFact
   INNER   JOIN CLIENŢI C ON         F.CodCl=C.CodCl
GROUP BYGROUPING SETS   ( (DenCl, DenPr),
      CAST( DataFact AS CHAR(14))      )
ORDER BY DenCl, DenPr, CAST (DataFact AS CHAR(14) )
```

în clauza GROUP BY a fost introdus operatorul GROUPING SETS, prin care se precizează două grupuri de agregare, unul alcătuit din combinaţia (DenCl, DenPr), celălalt din data facturii, astfel încât se obţine un rezultat cum este cel din figura 7.14.

Dacă se doreşte o situaţie sintetică cu subtotaluri ale celor trei atribute, plus un total general (obţinut prin perechea de paranteze fară argumente), se schimbă clauza GROUP BY astfel:

CLIENT	PRODUS	ZIUA	VINZARI
Clienţi SRL	Produs 1	y-subtotal-	3385550.00
Clienţi SRL	Produs 2	y-subtotal-	5237190.00
Clienţi SRL	Produs 5	y-subtotal-	3867500.00
Client 2 SA	Produs 2	y-subtotal-	1160250.00
Client 3 SRL	Produs 2	y-subtotal-	453985.00
Client 3 SRL	Produs 3	y-subtotal-	333200.00
Client 3 SRL	Produs 4	y-subtotal-	833000.00
Client 3 SRL	Produs 5	y-subtotal-	5622750.00
Client 4	Produs 2	y-subtotal-	1249500.00
Client 4	Produs 5	y-subtotal-	4188800.00
Client 5 SRL	Produs 2	y-subtotal-	980560.00
Client 5 SRL	Produs 3	y-subtotal-	357000.00
Client 6 SA	Produs 2	y-subtotal-	891310.00
Client 6 SA	Produs 4	y-subtotal-	564060.00
Client 6 SA	Produs 5	y-subtotal-	5331200.00
Client 7 SRL	Produs 2	y-subtotal-	1383375.00
y-subtotal-	y-subtotal-	2000-08-01	14684005.00
y-subtotal-	y-subtotal-	2000-08-02	3034500.00
y-subtotal-	y-subtotal-	2000-08-03	2320500.00
y-subtotal-	y-subtotal-	2000-08-04	2052750.00
y-subtotal-	v-subtotal-	2000-08-07	13747475.00

Figura 7.14. Clauza GROUPING SETS

```
SELECT
   CASE GROUPING (DenCl)
      WHEN 1 THEN CONCAT (CHR(255),    ' -subtotal- ')
      ELSE DenCl END AS Client,
   CASE GROUPING (DenPr)
      WHEN 1 THEN CONCAT (CHR(255),    ' -subtotal- ')
      ELSE DenPr END AS Produs,
   CASE GROUPING (CAST( DataFact AS CHAR(14) )         )
      WHEN 1 THEN CONCAT (CHR(255),    ' -subtotal- ')
      ELSE CAST( DataFact AS CHAR(14) ) END AS Ziua,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P
   INNER JOIN LINIIFACT LF ON P.CodPr=LF.Codpr INNER
JOIN FACTURI F ON LF.NrFact=F.NrFact INNER JOIN
CLIENŢI C ON F.CodCl=C.CodCl GROUP BY GROUPING SETS
(DenCl, DenPr,
   CAST ( DataFact AS CHAR(14) ),          ( ) )
ORDER BY DenCl, DenPr, CAST (DataFact AS CHAR(14) )
```

CLIENT	PRODUS	ZIUA	VINZARI
Clienţi SRL	y-subtotal-	y-subtotal-	12490240.00
Client 2 SA	y-subtotal-	y-subtotal-	1160250.00
Client 3 SRL	y-subtotal-	y-subtotal-	7242935.00
Client 4	y-subtotal-	y-subtotal-	5438300.00
Client 5 SRL	y-subtotal-	y-subtotal-	1337560.00
Client 6 SA	y-subtotal-	y-subtotal-	6786570.00
Client 7 SRL	y-subtotal-	y-subtotal-	1383375.00
y-subtotal-	Produs 1	y-subtotal-	3385550.00
y-subtotal-	Produs 2	y-subtotal-	11356170.00
y-subtotal-	Produs 3	y-subtotal-	690200.00
y-subtotal-	Produs 4	y-subtotal-	1397060.00
y-subtotal-	Produs 5	y-subtotal-	19010250.00
y-subtotal-	y-subtotal-	2000-08-01	14684005.00
y-subtotal-	y-subtotal-	2000-08-02	3034500.00
y-subtotal-	y-subtotal-	2000-08-03	2320500.00
y-subtotal-	y-subtotal-	2000-08-04	2052750.00
y-subtotal-	y-subtotal-	2000-08-07	13747475.00
y-subtotal-	y-subtotal-	y-subtotal-	35839230.00

Figura 7.15. GROUPING SETS - exemplul 2

Cele 18 linii din figura 7.15 sunt obţinute astfel: 7 clienţi pentru care sunt vânzări, plus 5 produse vândute, plus 5 zile în care s-au întocmit facturi, plus linia totalului general.

Interesant este şi faptul că cei trei operatori pot fi combinaţi. Spre exemplu, dacă GROUP BY are forma:

```
GROUP BY
GROUPING SETS ( ROLLUP (DenCl, DenPr),
      CAST( DataFact AS CHAR(14))       )
```

prin execuţia consultării se obţin liniile din figura 7.16.

CLIENT	PRODUS	ZIUA	VINZARI
Iclient 1 SRL	Produs 1	!y-subtotal-	3385550.00
Client 1 SRL	Produs 2	ly-subtotal-	5237190.00
IClient 1 SRL	Produs 5	iy-subtotal-	3867500.00
I Client 1 SRL	y-subtotal-	iy-subtotal-	12490240.00
IClient 2 SA	Produs 2	y-subtotal-	1160250.00
Client 2 SA	y-subtotal-	y-subtotal-	1160250.00
Client 3 SRL	Produs 2	y-subtotal-	453985.00
(Client 3 SRL	Produs 3	y-subtotal-	333200.00
IClient 3 SRL	Produs 4	y-subtotal-	833000.00
IClient 3 SRL	Produs 5	y-subtotal-	5622750.00
IClient 3 SRL	y-subtotal-	y-subtotal-	7242935.00
Client 4	Produs 2	y-subtotal-	1249500.00
Client 4	Produs 5	y-subtotal-	4188800.00
Client 4	y-subtotal-	y-subtotal-	5438300.00
Client 5 SRL	Produs 2	y-subtotal-	980560.00
Client 5 SRL	Produs 3	y-subtotal-	357000.00
Client 5 SRL	y-subtotal-	y-subtotal-	1337560.00
Client 6 SA	Produs 2	y-subtotal-	891310.00
Client 6 SA	Produs 4	y-subtotal-	564060.00
Client 6 SA	Produs 5	y-subtotal-	5331200.00
IClient 6 SA	y-subtotal-	y-subtotal-	6786570.00
Client 7 SRL	Produs 2	y-subtotal-	1383375.00
Client 7 SRL	y-subtotal-	y-subtotal-	1383375.00
y-subtotal-	y-subtotal-	2000-08-01	14684005.00
y-subtotal-	y-subtotal-	2000-08-02	3034500.00
y-subtotal-	y-subtotal-	2000-08-03	2320500.00
ly-subtotal-	y-subtotal-	(2000-08-04	2J52750.00
y-subtotal-	y-subtotal-	2000-08-07	13747475 00
iy-subtotal-	y-subtotal-	y-subtotal-	35839230.00

CUBE-uri parţiale

Discuţia se poartă în termeni similari ROLLUP-ului parţial. Subtotalurile şi combinaţiile posibile sunt limitate la atributele-dimensiuni incluse între paranteze. Spre exemplu, dacă
GROUP BY are forma:

```
GROUP BY atributl, CUBE (atribut2, atribut3)
```

vor fi calculate patru subtotaluri, pentru combinaţiile:

```
(atributl, atribut2, atribut3) - GROUP BY-ul obişnuit,
(atributl, atribut2),
(atributl, atribut3),
(atributl).
```

Versiune DB2:

IIKSI

```
SELECT
   CASE GROUPING (DenCl)
      WHEN 1 THEN CONCAT (CHR(255),    '  subtotal ' )
      ELSE DenCl END AS Client,
   CASE GROUPING (DenPr)
      WHEN 1 THEN CONCAT (CHR(255), 'subtotal ')
      ELSE DenPr END AS Produs,
   CASE GROUPING (CAST( DataFact AS CHAR(14) ) )
      WHEN 1 THEN CONCAT (CHR(255),    '  subtotal  ')
      ELSE CAST( DataFact AS CHAR(14) ) END AS Ziua,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari
FROM PRODUSE    P
   INNER JOIN   LINIIFACT  LF ON P.CodPr=LF.Codpr
   INNER JOIN   FACTURI  F ON LF.NrFact=F.NrFact
   INNER JOIN   CLIENŢI  C ON F.CodCl=C.CodCl
```

Figura 7.16. GROUPING SETS şi ROLLUP

```
GROUP BY DenCl, CUBE (DenPr, CAST( DataFact AS CHAR(14) ) ) ORDER BY
DenCl, DenPr, CAST (DataFact AS CHAR(14) )
```

Tabelul 7.2. CUBE parţial

CLIENT	PRODUS	ZIUA	VINZARI
Client 1 SRL	Produs 1	01/08/2000	595000
Client 1 SRL	Produs 1	03/08/2000	1130500
Client 1 SRL	Produs 1	04/08/2000	1660050
Client 1 SRL	Produs 1	y subtotal	3385550
Client 1 SRL	Produs 2	01/08/2000	937125
Client 1 SRL	Produs 2	02/08/2000	1651125
Client 1 SRL	Produs 2	03/08/2000	1190000
Client 1 SRL	Produs 2	04/08/2000	392700
Client 1 SRL	Produs 2	07/08/2000	1066240
Client 1 SRL	Produs 2	y subtotal	5237190
Client 1 SRL	Produs 5	01/08/2000	3867500
Client 1 SRL	Produs 5	y subtotal	3867500
Client 7 SRL	Produs 2	02/08/2000	1383375
Client 7 SRL	ysubtotai	02/08/2000	1383375
y subtotal	Produs 1	01/08/2000	595000
y subtotal	Produs 1	03/08/2000	1130500
y subtotal	Produs 1	04/08/2000	1660050
y subtotal	Produs 2	01/08/2000	3969245
y subtotal	Produs 2	02/08/2000	3034500
y subtotal	Produs 2	03/08/2000	1190000
y subtotal	Produs 2	04/08/2000	392700
y subtotal	Produs 2	07/08/2000	2769725
y subtotal	Produs 3	01/08/2000	357000
y subtotal	Produs 3	07/08/2000	333200
y subtotal	Produs 4	01/08/2000	564060
y subtotal	Produs 4	07/08/2000	833000
y subtotal	Produs 5	01/08/2000	9198700
y subtotal	Produs 5	07/08/2000	9811550
y subtotal	ysubtotai	01/08/2000	14684005
y subtotal	ysubtotai	02/08/2000	3034500
y subtotal	ysubtotai	03/08/2000	2320500
y subtotal	ysubtotai	04/08/2000	2052750
y subtotal	ysubtotai	07/08/2000	13747475

7.3. Clasamente - soluţii clasice şi OLAP

Prefaţăm discuţia despre funcţiile *analitice* din SQL-99 cu o problemă mai veche decât SQL-ul -

clasamentele.

Să se afişeze clasamentul zilelor în ordinea descrescătoare a numărului de facturi emise.

Gradul de dificultate al interogărilor care să rezolve această problemă depinde de modul în care se doreşte a se obţine clasamentul. Cine se mulţumeşte cu lista din figura 7.17 poate folosi câteva interogări relativ simple.

DATAFACT	NR FACTURILOR
2000-08-01	4
2000-08-07	4
2000-08-02	2
2000-08-03	1
2000-08-04	1

O variantă comună Oracle/DB2, ba chiar şi VFP, este:

```
SELECT DataFact, COUNT(*) AS Nr_Facturilor
FROM FACTURI
GROUP BY DataFact
ORDER BY Nr_Facturilor DESC
```

Ba `chiar` în Oracle se poate folosi şi o subconsultare în clauza `FROM`:

```
SELECT DataFact, Nr_Facturilor
FROM
   (SELECT DataFact, COUNT(*) AS Nr_Facturilor
   FROM FACTURI
   GROUP BY DataFact )
ORDER BY Nr_Facturilor DESC
```

iar în DB2 o expresie-tabelă:

```
WITH ZILE_FACT AS
   (SELECT DataFact, COUNT(*) AS Nr_Facturilor
   FROM FACTURI GROUP BY DataFact)
SELECT *
FROM ZILE_FACT
ORDER BY Nr_Facturilor DESC
```

Ce ne facem însă atunci când rezultatul trebuie să îmbrace forma din figura 7.18? Problema esenţială este nu ordonarea, cât indicarea poziţiei fiecărei zile.

DATAFACT	NR_FACTURILOR	POZIŢIE
2000-08-01	4	1
2000-08-07	4	1
2000-08-02	2	3
2000-08-03	1	4
2000-08-04	1	4

Figura 7.18. Un clasament veritabil

în Oracle 8i2 este posibilă ordonarea liniilor unei subconsultări din clauza `FROM`, aşa că soluţia următoare, bazată pe pseudocoloana `ROWNUM`, funcţionează:

```
SELECT DataFact, Nr_Facturilor, ROWNUM AS Poziţie
FROM
   (SELECT DataFact, COUNT(*) AS Nr_Facturilor
   FROM FACTURI
   GROUP BY DataFact
```

```
ORDER BY Nr_Facturilor DESC)
```

Chiar dacă ne putem declara satisfăcuţi de rezultat (figura 7.19), se poate argumenta că 7 august 2000 este pe nedrept „retrogradată" pe locul 2 al clasamentului, atât timp cât are acelaşi număr de zile ca şi 1 august şi nu există nici un criteriu suplimentar de departajare. Interogarea de mai sus este corectă sută la sută atunci când fie toate valorile ordonate sunt diferite, fie când există un număr suficient de criterii de balotaj care să confere unicitate fiecărei poziţii a clasamentului.

DATAFACT	NR_FACTURILOR	POZIŢIE
2000-08-01	4	1
2000-08-07	4	2
2000-08-02	2	3
2000-08-03	1	4
2000-08-04	1	5

Figura 7.19. Pseudoclasamentul obţinut printr-o subconsultare

Pentru astfel de situaţii, Amendamentul OLAP al SQL-99 prezintă două funcţii ideale, `RANK` şi `DENSE_RANK`. în prealabil însă, câteva chestiuni ce ţin de logica execuţiei unui asemenea gen de consultări. Procesarea interogărilor ce utilizează funcţii analitice se derulează în trei etape. Mai întâi, se operează toate joncţiunile, se constituie grupurile şi se efectuează selecţia asupra grupurilor, altfel spus, se execută clauzele `JOIN` (sau `FROM/WHERE), WHERE, GROUP BY` şi `HAVING`. Apoi se aranjează rezultatul în vederea aplicării funcţiilor analitice şi se efectuează calculele (sunt create partiţiile), iar funcţiile analitice sunt aplicate linie cu linie în fiecare partiţie. Pasul 3 este operaţional numai dacă interogarea prezintă la sfârşit o clauză `ORDER BY,` ceea ce atrage ordonarea finală a rezultatului în conformitate cu criteriile specificate.

Partiţiile sunt seturi de linii create după delimitarea grupurilor prin `GROUP BY,` astfel încât pot constitui subiectul (sau obiectul) oricărei funcţii de agregare `(SUM, AVG....).` Constituirea unei partiţii se poate face în funcţie de valorile unuia sau mai multor atribute sau expresii de atribute.

Soluţia comună Oracle 8i2/DB2 v7 pentru a obţine o situaţie identică celei din figura 7.18 este:

```
SELECT DataFact/ COUNT(*) AS Nr_Facturilor,
   RANK() OVER (ORDER BY COUNT(*) DESC) AS Poziţie FROM
FACTURI GROUP BY DataFact
```

Funcţia `RANK` ierarhizează după valorile obţinute de `COUNT` (*) liniile rezultatului şi, ceea ce este cel mai important, la valori egale, atribuie fiecărei linii aceeaşi poziţie în clasament.

Luăm un alt exemplu care să pună în valoare şi partiţii.

Să se afişeze, pentru fiecare zi de vânzări, topul celor mai „ valoroase "facturi.

Să analizăm problema:

- partiţie are dimensiunea unei zile; numărul liniilor depinde de numărul facturilor întocmite în ziua respectivă; prin urmare, atributul de partiţionare este `DataFact;`
- în cadrul unei partiţii, clasamentul se face în funcţie de valorile calculate de funcţia SUM;
- în rezultatul final, pentru fiecare factură, liniile sunt dispuse în funcţie de valorile DataFact şi NrFact, deci la prezentare se păstrează ordinea cronologică.

Iată şi soluţia DB2 v7 şi Oracle 8i2 (rezultatul în figura 7.20):

```
SELECT DataFact, F.NrFact,
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValFact,
   RANK() OVER (PARTITION BY DataFact ORDER BY
   SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
   AS Poziţie
```

```
FROM PRODUSE P, LINIIFACT LF, FACTURI F WHERE
P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact GROUP BY
DataFact, F.NrFact ORDER BY DataFact, F.NrFact
```

DATAFACT	NRFACT	VALFACT	POZIŢIE
2000-08-01	1111	5399625.00	2
12000-08-01	1112	1337560.00	3
2000-08-01	1113	1160250.00	4
2000-08-01	1114	6786570.00	1
2000-08-02	1115	1651125.00	1
2000-08-02	1116	1383375.00	2
2000-08-03	1117	2320500.00	1
2000-08-04	1118	2052750.00	1
2000-08-07	1119	7242935.00	1
2000-08-07	1120	1066240.00	3
2000-08-07	1121	5438300.00	2

Figura 7.20. Topurile zilnice ale facturilor

Factura 1111 este a doua ca valoare pe 1 august 2000, cea mai mare valoare în această zi având factura 1114.

Intr-o singură frază SELECT se pot întocmi, pentru aceleaşi date, clasamente diferite prin specificarea mai multor partiţii şi criterii. Astfel, interogareâ (Oracle 8i2/DB2 v7):

```
SELECT DataFact, DenPr,
         SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValFact,
   RANK() OVER (PARTITION BY DataFact ORDER BY
            SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
      AS Pozitie__Zi,
   RANK() OVER (PARTITION BY DenPr ORDER BY
            SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
      AS Pozitie_Prod FROM PRODUSE P, LINIIFACT LF,
FACTURI F WHERE P.CodPr=LF.CodPr AND
LF.NrFact=F.NrFact GROUP BY DataFact, DenPr ORDER BY
DataFact, DenPr
```

calculează valoarea vânzărilor pe zile `(DataFact)` pentru fiecare produs şi, în plus, determină poziţia produsului `(DenPr)` curent, atât pentru ziua respectivă, cât şi pe ansamblu, pentru toate combinaţiile zi-produs `(DataFact-DenPr)`, după cum se observă în figura 7.21.

DATAFACT	DENPR	VALFACT	POZIŢIE_ZI	POZITIE_PROD
2000-08-01	Produs 1	595000.00	3	3
2000-08-01	Produs 2	3969245.00	2	1
2000-08-01	Produs 3	357000.00	5	1
2000-08-01	Produs 4	564060.00	4	2
2000-08-01	Produs 5	91 98700.00	1	2
2000-08-02	Produs 2	3034500.00	1	2
2000-08-03	Produs 1	L 1130500.00	2	2
2000-08-03	Produs 2	11 90000.00	1	4
2000-08-04	Produs 1	1 660050.00	1	1
2000-08-04	Produs 2	392700,00	2	5
2000-08-07	Produs 2	2769725.00	2	3
2000-08-07	Produs 3	333200.00	4	2
2000-08-07	Produs 4	833000.00	3	1
2000-08-07	Produs 5	9811550.00	1	1

Pe prima linie a rezultatului apar ziua de *1 august 2000* şi *produsul 1*, pentru care valoarea facturată în această zi este de 595.000. 595.000 este, ca mărime, a treia valoare pentru ziua respectivă şi a treia pentru produsul 1. Altfel spus, pentru 1 august 2000, produsul 1 este al treilea în topul pe produse al vânzărilor (cel mai bine vândut în această zi fiind produsul 5), iar dintre toate zilele în care a fost vândut „Produs 1”, 1 august este a treia în topul vânzărilor pe zile (cel mai bine s-a vândut pe 4 august).

„Produs 2” s-a vândut cel mai bine pe 1 august şi cel mai puţin pe 4 august; pe 2 şi pe 3 august a fost cel mai bine

Figura 7.21. Două partiţii de ordonare

vândut produs al zilei.

Ziua de graţie a produsului 5 a fost 7 august, când a fost cel mai bine vândut produs al zilei şi data cu cele mai mari vânzări ale acestuia.

Se poate alcătui un clasament nu numai pe baza unei simple funcţii-agregat, dar şi prin combinarea funcţiei RANK cu funcţiile ROLLUP şi CUBE. Rezultatele nu sunt însă întotdeauna concludente. Spre exemplu, consultarea DB2 v7 următoare produce raportul din figura 7.22.

```
SELECT Judeţ, DenPr,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari,
   RANK() OVER (PARTITION BY Judeţ
     ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC) AS
   Poz_Judet,
   RANK() OVER (PARTITION BY DenPr
     ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC) AS
Poz_Produs FROM PRODUSE  P
   INNER   JOIN LINIIFACT  LF ON  P.CodPr=LF.Codpr
   INNER   JOIN FACTURI  F   ON LF.NrFact=F.NrFact
   INNER   JOIN CLIENŢI  C   ON F.CodCl=C.CodCI
   INNER   JOIN LOCALITATI  L  ON  C.CodPost=L.CodPost
   INNER JOIN JUDEŢE J ON L.Jud=J.Jud
GROUP BY ROLLUP (Judeţ, DenPr)
ORDER BY Judeţ, DenPr
```

JUDEŢ	DENPR	VINZARI	POZ_JUDET	POZ_PRODUS
Iasi	Produs 1	3385550.00	4	1
Iasi	Produs 2	7646940.00	3	1
Iasi	Produs 5	8056300.00	2	1
Iasi		19088790.00	1	2
Neamţ	Produs 2	891310.00	3	3
Neamţ	Produs 4	564060.00	4	2
Neamţ	Produs 5	5331200.00	2	3
Neamţ		6786570.00	1	4
Timiş	Produs 2	2363935.00	2	2
Timiş	Produs 3	357000.00	3	1
Tirnis		2720935.00	1	5
Vaslui	Produs 2	453985.00	4	4
Vaslui	Produs 3	333200.00	5	2
Vaslui	Produs 4	833000.00	3	1
Vaslui	Produs 5	5622750.00	2[1]	2
Vaslui		7242935.00	1	3
		35839230.00	1	1

Figura 7.22. RANK şi ROLLUP

Este limpede că Poz_Judet va fi întotdeauna 1 în liniile dedicate subtotalului pentru fiecare dintre judeţe. „Produs 1" nu este al patrulea ca vânzări în judeţul Iaşi, ci al treilea. Valorile Poz_Produs sunt însă coerente. Judeţul în care „Produs 4" s-a vândut cel mai bine este Vaslui.

Cea mai importantă informaţie ţine de faptul că, dacă luăm în calcul liniile subtotalurilor şi totalului general, aflăm, spre exemplu, din penultima linie, că judeţul Vaslui este al treilea ca volum total al vânzărilor. Poziţia exactă a Vasluiului în topul judeţelor este însă a doua, doarece prima îi revine automat totalului general (ultima linie).

Lucrurile stau aidoma şi în cazul operatorului CUBE. Deşi raportul (figura 7.23) furnizat de următoarea frază SELECT, valabilă deopotrivă în Oracle 8i2 şi DB2 v7, este bogat în informaţii, câteva dintre acestea trebuie nuanţate.

```
SELECT
  CASE WHEN GROUPING (DenCl) = 1
    THEN CONCAT (CHR(255),   '  - SUBTOTAL - ')
   ELSE DenCl END AS
```

Figura 7.23. RANK şi CUBE

```
            Client,
   CASE WHEN GROUPING (DenPr) = 1
      THEN CONCAT (CHR(255),   '  - SUBTOTAL - ')
     ELSE DenPr END AS
            Produs,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari,
      RANK() OVER (PARTITION BY DenCl
      ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC) AS
      Poz_Client,
   RANK() OVER (PARTITION BY DenPr
      ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC) AS
Poz_Produs FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENŢI C,
LOCALITATI L, JUDEŢE J WHERE P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact
AND
   F.CodCl=C.CodCI AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP BY
CUBE (DenCl, DenPr )
ORDER BY DenCl, DenPr
```

CLIENT	PRODUS	VINZARI	POZ_CLIENT	POZ PRODUS
Clienţi SRL	Produs 1	3385550.00	4	1
Clienţi SRL	Produs 2	5237190.00	2	2
Clienţi SRL	Produs 5	3867500.00	3	5
Clienţi SRL	y-SUBTOTAL-	12490240 00	1	2
Client 2 SA	Produs 2	1160250.00	1	5
Client 2 SA	y-SUBTOTAL-	1160250.00	1	8
Client 3 SRL	Produs 2	453985.00	4	8
Client 3 SRL	Produs 3	333200 00	5	3
Client 3 SRL	Produs 4	833000 00	3	2
Client 3 SRL	Produs 5	5622750 00	2	2
Client 3 SRL	y- SUBTOTAL-	7242335.00	1	3
Client 4	Produs 2	12495C0 00	3	4
Client 4	Produs 5	4183800 00		4
Client 4	V- SUBTOTAL-	5438300 00	1	5
Client 5 SRL	Produs 2	930560 00	2	6
Client 5 SRL	Produs 3	357000.00	3	2
Client 5 SRL	y-SUBTOTAL-	1337560.00	1	7
Client 6 SA	Produs 2	891310.00	3	7
Client 6 SA	Produs 4	564060.00	4	3
Client 6 SA	Produs 5	5331200.00	2	3
Client 6 SA	y- S UBTOTAL -	6786570.00	1	4
Client 7 SRL	Produs 2	1383375.00	1	3
Client 7 SRL	y-SUBTOTAL-	1383375 00	1	6
y-SUBTOTAL-	Produs 1	3385550.00	4	1
y- SUBTOTAL•	Produs 2	11356170.00	3	1
y-SUBTOTAL-	Produs 3	690200.00	6	1
y-SUBTOTAL-	Produs 4	1397060.00	5	1
y-SUBTOTAL-	Produs 5	19010250.00	2	1
V- SUBTOTAL -	V-SUBTOTAL- 1	35839230.00	1	1

Astfel, ca vânzări, pentru *Client 1 SRL, Produs 2* este pe locul 1, şi nu pe locul 2, cum indică a doua linie din figură. La modul general, `Poz_Client` este tot timpul „umflat" cu o poziţie datorită subtotalurilor de la nivel de client şi, pentru ultimele linii, a totalului general. Analog, capul listei pentru `Poz_Produs` este cel al subtotalului pe produs, iar, spre exemplu, cel mai bun cumpărător al *Produs 2* este *Client 1 SRL*. Faptul că, la *Produs 1,* în *dreptul Clientului 1 SRL* este indicată poziţia corectă, acest lucru se datorează vânzării produsului unui singur client.

Ţinând seama de interferenţa subtotalurilor, am fi tentaţi să decrementăm rezultatele funcţiilor `RANK` cu o unitate:

```
SELECT
   CASE WHEN GROUPING (DenCl) = 1
      THEN CONCAT (CHR(255),   '   -     SUBTOTAL - ')
     ELSE DenCl END AS
            Client,
   CASE WHEN GROUPING (DenPr) = 1
      THEN CONCAT (CHR(255),   '   -     SUBTOTAL - ')
      ELSE DenPr       •
```

```
   END AS Produs,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari,
   RANK() OVER (PARTITION BY DenCl ORDER BY
      SUM(Cantitate * PretUnit     *              (1+ProcTVA))
                                   DESC)          -       1
      AS Poz_Client,
   RANK() OVER (PARTITION BY DenPrORDER BY
      SUM(Cantitate * PretUnit     *              (1+ProcTVA))
                                   DESC)          -       1
   AS Poz_Produs FROM PRODUSE P, LINIIFACT LF, FACTURI F,
CLIENŢI C, LOCALITATI L, JUDEŢE J WHERE P.CodPr=LF.Codpr AND
LF.NrFact=F.NrFact AND
   F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP BY
CUBE (DenCl, DenPr )
ORDER BY DenCl, DenPr
```

Rezultatele ar fi perfecte dacă n-ar exista produse vândute unui singur client (*Produs 1*) şi clienţi care au cumpărat un singur produs (*Client 2* şi *Client* 7). Pentru aceste două speţe, `Poz_Client`, respectiv `Poz_Produs` sunt 0, chiar dacă respectivul produs/client ar fi singur şi poziţia ar trebui să fie, inevitabil, 1. Am modificat, astfel, gluma aceea răsuflată: să alergi de unul singur şi ajungi pe locul zero!

O soluţie mult mai elegantă se bazează pe modificarea modului de declarare a partiţiilor. Pentru a pune în evidenţă cele două modalităţi, se afişează câte două poziţii pentru clienţi/produse, folosindu-se câte două funcţii `RANK`; una identică cu exemplul anterior şi o alta care beneficiază de serviciile `GROUPING`, după cum urmează:

```
SELECT
   CASE WHEN GROUPING (DenCl) = 1
      THEN  CONCAT  (CHR(255),  ' - SUBTOTAL   -  ')
      ELSE  DenCl                                  .
        END AS Client,
   CASE WHEN GROUPING (DenPr) = 1
      THEN  CONCAT  (CHR(255),  ' - SUBTOTAL   -  ')
    ELSE DenPr END AS
              Produs,
```

Figura 7.23. RANK şi CUBE

```
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari,
      RANK() OVER (PARTITION BY DenCl
      ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC) AS
      Poz_Client_1,
   RANK() OVER (PARTITION BY DenCl, GROUPING(DenPr)
      ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC) AS
      Poz_Client_2,
   RANK() OVER (PARTITION BY DenPr
      ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC) AS
      Poz_Produs_1,
   RANK() OVER (PARTITION BY DenPr, GROUPING (DenCl)
      ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC) AS
Poz_Produs_2 FROM PRODUSE P, LINIIFACT LF, FACTURI F, CLIENŢI C,
LOCALITATI L, JUDEŢE J WHERE P.CodPr=LF.Codpr AND
LF.NrFact=F.NrFact AND
   F.CodCl=C.CodCl AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP
BY CUBE (DenCl, DenPr )
ORDER BY GROUPING(DenCl), DenCl, GROUPING(DenPr), DenPr
```

Coloanele `Poz_Client_2` şi `Poz_Produs_2` conţin, graţie includerii clauzelor `GROUPING` în funcţiile `RANK`, valorile care ne interesează - vezi figura 7.24.

CLIENT	PRODUS	VINZARI	POZ.CLIENTJ	POZ_CLIENT_2	POZ_PRODUS_1	POZ_PRODUS_2
Clienţi SRL	Produs 1	3385550 00	4	3	1	1
Clienţi SRL	Produs 2	5237190 00	2,	1	2	1
Clienţi SRL	Produs 5	3867500.00	3	2	5	4
Clienţi SRL	y- SUBTOTAL-	12490240.0 0	1	1,	2	1
Client 2 SA	Produs 2	1160250.00	1	1	5	4
Client 2 SA	y-SUBTOTAL-	1160250.00	1	1	8	7
Client 3 SRL	Produs 2	453985.00	4	3	8	7
Client 3 SRL	Produs 3	333200.00	5	4	3	2
Client 3 SRL	Produs 4	833000.00	3	2	2	1
Client 3 SRL	Produs 5	5 62 27 50 00	2	1	2	1
Client 3 SRL	y-SUBTOTAL-	7242935.00	1	1	3	2
Client 4	Produs 2	1249500.00	3	2	4	3
Client 4	Produs 5	4188800.00	2	1	4	3
Client 4	y-SUBTOTAL-	5438300.00	1	1	5	4
Client 5 SRL	Produs 2	980560.00	2	1	6	5
Client 5 SRL	Produs 3	357000.00	3	2	2	
Client 5 SRL	y- SUBTOTAL-	1337560.00	1	1	7	
Client 6 SA	Produs 2	891310.00	3	2	7	6
Client 6 SA	Produs 4	564060.00	4	3	3	2
Client 6 SA	Produs 5	5331200.00	2	1	3	2
Client 6 SA	y-SUBTOTAL-	67 86 570 00	1	1	4	3
Client 7 SRL	Produs 2	1383375.00	1	1	3	2
Client 7 SRL	y- SUBTOTAL-	1383375.00	1	1	6	5
y-SUBTOTAL-	Produs 1	3385550.00	4	3	1	1
y-SUBTOTAL-	Produs 2	11356170.0 0	3	2	1	1
y-SUBTOTAL-	Produs 3	690200.00	6	5	1	1
y-SUBTOTAL-	Produs 3	1397060.00	5	4	1	1
y-SUBTOTAL-	Produs 5	19010250.0 0	2	1	1	1
y-SUBTOTAL- '	y-SUBTOTAL -	35839230.0 0	1	1	1	1

Figura 7.24. CUBE şi RANK, cu şi fără GROUPING

Care sunt cele mai mari cinci preţuri unitare din LINIIFACT?

In paragraful 5.4 am prezentat o soluţie ingenioasă bazată pe subconsultări în cascadă. O dată cu versiunea 8i, Oracle suportă ordonări în subconsultări, astfel încât următoarea frază este operaţională:

```
SELECT PretUnit
FROM
   (SELECT PretUnit FROM
LINIIFACT ORDER BY PretUnit DESC)
PI WHERE ROWNUM <=5
```

Din păcate, în DB2, pseudocoloana `ROWNUM` nu este operaţională. în schimb, versiunile 7 şi 8i2 ale DB2 şi Oracle implementează o nouă funcţie OLAP din SQL-99, funcţie cu o logică similară, `ROW_NUM`, astfel încât se poate redacta o soluţie comună de genul:

```
SELECT PretUnit
FROM
   (SELECT PretUnit, ROW_NUMBER () OVER
(ORDER BY PretUnit DESC) AS Poz FROM
LINIIFACT ) PI WHERE Poz <=5
```

Funcţia `ROW__NUMBER` întoarce numărul de ordine al fiecărei linii în partiţia declarată.

Care sunt cele mai mari cinci preţuri unitare de vânzare, produsele şi facturile în care apar cele cinci preţuri maxime?

Bazându-ne pe soluţiile anterioare, se poate redacta fraza următoare în sintaxa Oracle 8i2:

```
SELECT DISTINCT NrFact, DenPr, LF.PretUnit FROM
LINIIFACT LF, PRODUSE P,
   (SELECT PretUnit FROM
        (SELECT PretUnit
        FROM LINIIFACT
        ORDER BY PretUnit DESC)
   WHERE ROWNUM <=5 ) PMAX WHERE LF.CodPr = P.CodPr AND
LF.PretUnit >= PMAX.PretUnit ORDER BY LF.PretUnit DESC
```

şi o alta comună Oracle 8i2/DB2 v7:

```
SELECT DISTINCT NrFact, DenPr, LF.PretUnit FROM
LINIIFACT LF, PRODUSE P,
   (SELECT Poz, PretUnit FROM
     (SELECT PretUnit, ROW_NUMBER () OVER
     (ORDER BY PretUnit DESC) AS Poz FROM
     LINIIFACT ) PI
   WHERE Poz <=5 )
WHERE LF.CodPr = P.CodPr AND LF.PretUnit >= PMAX.PretUnit ORDER
BY LF.PretUnit DESC
```

Să se afişeze ziua sau zilele în care s-au emis cele mai multe facturi.

Interogarea Oracle 8i2:

```
SELECT DataFact, Nr__Facturilor, ROWNUM AS Poziţie FROM
   (SELECT DataFact, COUNT(*) AS Nr_Facturilor
   FROM FACTURI
   GROUP BY DataFact
   ORDER BY Nr_Facturilor DESC)
WHERE ROWNUM = 1
```

este incorectă, deoarece extrage numai una dintre cele două zile (1 şi 7 august) în care s-au emis câte patru facturi. Cu toate acestea, rezultatul corect poate fi obţinut şi fară a apela la funcţia RANK, ci prin folosirea abuzivă a subconsultărilor şi a posibilităţii versiunii 8i2 de a ordona liniile oricărei subconsultări:

```
SELECT ROWNUM AS Poziţie,
   TO_CHAR(DataFact,'DD-MM-YYYY') AS Zi, Nr_Facturilor FROM
   (SELECT DataFact, COUNT(*) AS Nr_Facturilor FROM FACTURI GROUP
BY DataFact HAVING COUNT(*) >= ANY (SELECT Nr_Facturilor FROM
         (SELECT DISTINCT COUNT(*) AS Nr_Facturilor
         FROM FACTURI
         GROUP BY DataFact
         ORDER BY Nr_Facturilor DESC)
      WHERE ROWNUM <= 1 )
   ORDER BY Nr_Facturilor DESC )
```

Rezultatul din figura 7.25 este satisfăcător, chiar dacă poziţia a 2-a în clasament a datei de 7 august este discutabilă, deoarece, de fapt, aceasta este pe primul loc, împreună cu 1 august.

POZIŢIE	ZI	NR_FACTORILOR
1	01-08-2000	4
2	07-08-2000	4

Figura 7.25. Zilele pentru care se înregistrează cel mai mare număr de facturi emise

Interogarea comună bazată pe funcţia analitică `ROW_NUMBER` care se materializează în acelaşi rezultat de mai sus este:

```
SELECT ROW_NUMBER() OVER (ORDER BY Nr_Facturilor DESC)
   AS Poziţie, DataFact AS Zi, Nr_Facturilor FROM
   (
   SELECT DataFact, COUNT(*) AS Nr_Facturilor FROM FACTURI GROUP
   BY DataFact
   HAVING COUNT{*) >= ANY (
      SELECT Nr_Facturilor
      FROM (
        SELECT Nr_Facturilor, ROW_NUMBER()
          OVER (ORDER BY Nr_Facturilor DESC) AS Poz
        FROM
          (- SELECT DISTINCT COUNT(*) AS Nr Facturilor FROM
          FACTURI                          "
          GROUP BY DataFact
        ) XI ) X2
      WHERE X2.Poz <=1 )
   ) X3
ORDER BY- Nr_Facturilor DESC
```

Folosind funcţia `RANK`, ambele zile au acelaşi număr în `top`, 1. Şi noua variantă este valabilă atât în Oracle 8i2, cât şi în DB2 v7.

```
SELECT *
FROM
   (SELECT RANK() OVER (ORDER BY COUNT(*) DESC) AS Poziţie,
DataFact AS Zi, COUNT(*) AS Nr_Facturilor FROM FACTURI GROUP BY
DataFact) X WHERE Poziţie <= 1
```

Să se afişeze cele mai mari două valori ale numărului zilnic de facturi emise, precum şi datele în care s-au înregistrat respectivele valori.

Varianta non-OLAP în Oracle 8i2 este:

```
SELECT ROWNUM AS Poziţie,
   TO_CHAR ( DataFact, 'DD-MM-YYYY' ) AS Zi, Nr_Facturilor FROM
   (SELECT DataFact, COUNT(*) AS Nr Facturilor FROM FACTURI
GROUP BY DataFact HAVING COUNT(*) >= ANY (SELECT Nr_Facturilor
FROM
        (SELECT DISTINCT COUNT(*) AS Nr Facturilor
        FROM FACTURI
        GROUP BY DataFact                    ■
        ORDER BY Nr_Facturilor DESC)
      WHERE ROWNUM <= 2 )
   ORDER BY Nr_Facturilor DESC )
```

POZIŢIE	zi NR_FACTURILOR
1	01-08-2000 4
2	07-08-2000 4
3	02-08-2000 2

Figura 7.26. Primele două valori ale numărului zilnic de facturi emise şi datele respective

Acelaşi rezultat se obţine modificând, în ultima interogare ce utilizează funcţia `ROW_NUMBER`, doar 1 cu 2, adică, în loc de `X2 . Poz <=1, X2 . Poz <=2.` Deci putem vorbi de o cvasigene- ralizare a acestei interogări.

Obţinerea poziţiei corecte în rezultat se plăteşte scump, dar merită:

```
SELECT RANK() OVER (ORDER BY COUNT(*) DESC) AS Poziţie,
DataFact AS Zi, COUNT(*) AS Nr_Facturilor FROM FACTURI GROUP
BY DataFact HAVING COUNT(*) >= ANY
   ('
   SELECT Nr_Facturilor FROM
        (SELECT RANK() OVER
         (ORDER BY Nr_Facturilor DESC)
        AS Poz, Nr_Facturilor
   FROM
         (SELECT DISTINCT COUNT(*) AS Nr_Facturilor
   FROM FACTURI GROUP BY DataFact) XI ) X2 WHERE Poz
   <= 2 )
```

POZIŢIE	ZI	NR_FACTURILOR
1	2000-08-01	4
1	2000-08-07	4
3	2000-08-02	2

Figura 7.27. Indicarea corectă a poziţiei din clasament

Care sunt cele mai bine vândute două produse ale fiecărei zile?
Practic, interesează o listă precum cea din figura 7.28.

Raportul obţinut este cel din figura 7.26.

DATAFACT	DENPR	VINZARI	POZITIE_ZI
2000-08-01	Produs 5	9198700.00	1
2000-08-01	Produs 2	3969245.00	2
2000-08-02	Produs 2	3034500.00	1
2000-08-03	Produs 2	1190000.00	1
2000-08-03	Produs 1	1130500.00	2
2000-08-04	Produs 1	1660050.00	1
2000-08-04	Produs 2	392700.00	2
2000-08-07	Produs 5	9811550.00	1
2000-08-07	Produs 2	2769725.00	• 2

Figura 7.28. Cele mai bine vândute două produse din fiecare zi

Soluţia următoare este una comună DB2 v7/Oracle 8i2:

```
SELECT *
FROM
   (SELECT DataFact, DenPr, SUM(Cantitate * PretUnit *
       (1+ProcTVA)) AS Vinzari,
     RANK() OVER (PARTITION BY DataFact
     ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC) AS
Pozitie_Zi FROM PRODUSE P, LINIIFACT LF, FACTURI F WHERE
P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact GROUP BY DataFact,
DenPr ) XI WHERE Pozitie_Zi <= 2
```

Asemănătoare funcţiei RANK este DENSE_RANK, care, spre deosebire de prima, nu lasă goluri în clasament atunci când pe poziţia precedentă sunt două sau mai multe linii. Diferenţa este pusă în valoare de interogarea următoare şi rezultatul acesteia din figura 7.29.

```
SELECT DataFact, COUNT(*) AS Nr Facturilor,
   RANK() OVER (ORDER BY COUNT(*) DESC) AS Poz_RANK,
   DENSE_RANK() OVER (ORDER BY COUNT(*) DESC)
     AS Poz_DENSE_RANK FROM FACTURI                    .
GROUP BY DataFact
```

DATAFACT	NR_FACTURILOR	POZ_RANK	POZ_DENSE_RANK
2000-08-01	4 1 1 4 1 1		
2000-08-07	2 3 2		
2000-08-02			
2000-08-03	1 4		3
2000-08-04	1 4		3

Figura 7.29. Diferenţa dintre RANK şi DENSE_RANK

Cu DENSE_RANK se mai simplifică şi interogarea ce afişează cele mai mari două valori ale numărului zilnic de facturi emise, precum şi datele (calendaristice) respective:

```
SELECT DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS Poziţie,
DataFact AS Zi, COUNT (*) AS Nr__Facturilor FROM FACTURI GROUP
BY DataFact HAVING COUNT(*) >= ANY (
     SELECT Nr_Facturilor FROM
     (SELECT DENSE_RANK() OVER (ORDER BY COUNT(*) DESC)
       AS Poz, COUNT(*) AS Nr_Facturilor
   FROM FACTURI GROUP BY DataFact ) X2
   WHERE Poz <= 2 )
```

POZIŢIE	ZI	NR_FACTURILOR
1	2000-08-01	4
1	2000-08-07	4
2	2000-08-02	2

Figura 7.30. DENSE_RANK pentru obţinerea primelor două valori ale numărului zilnic de facturi

în afara folositelor ROW_NUMBER, RANK şi DENSE__RANK, Oracle 8i2 a fost gândit să-şi apropie şi mai mult o categorie profesională sedusă, de mult timp, de programe precum SPSS sau chiar Excel, şi anume statisticienii. Apar astfel funcţiile: CUME_DIST, PERCENT_RANK, NTILE. Pentru a le compara, folosim interogarea următoare, al cărei rezultat este vizualizat în figura 7.31.

```
SELECT DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS Poziţie, TO_CHAR(DataFact,'YYYY-
    MM-DD') AS Zi,
        COUNT(*) AS Nr_Facturilor,
    PERCENT_RANK() OVER (ORDER BY COUNT(*) DESC )
        AS Poz_Proc_Zi,
    CUME_DIST() OVER (ORDER BY COUNT(*) DESC )
        AS Distrib_Cum_Zi,
    NTILE(3) OVER (ORDER BY COUNT(*) DESC ) AS Tertile_Zi, NTILE(4) OVER (ORDER BY
COUNT(*) DESC ) AS Cuartile_Zi FROM FACTURI GROUP BY DataFact
```

POZIŢIE	ZI	NR_FACTURILOR	POZ_PROC_ZI	DISTRIB_CUM_ZI	TERTILE_ZI	CUARTILE_ZI
1	2000-08-01	4	0	.4	1	1
1	2000-08-07	4	0	.4	1	1
2	2000-08-02	2	.5	.6	2	2
3	2000-08-03	1	.75	1	2	3
3	2000-08-04	1	.75	1	3	4

Figura 7.31. Funcţiile Oracle PERCENT_RANK, CUME_DIST şi NTILE

PERCENT RANK întoarce poziţia procentuală a unei linii în cadrul unei partiţii, calculată prin formula: *(poziţia liniei în partiţie* - 1) / *{nr. liniilor din partiţie - 1).*

CUME_DIST calculează poziţia unei valori specificate x relativă la o mulţime *M* de *N* valori: CUME DI ST (x) = numărul de valori (diferite de, sau egale cu x) din M ce precedă pe *x* (în ordinea specificată) / N.

NTILE (3) calculeazătertilele, iar NTILE (4) calculeazăcuartilele.

7.4. Ferestre pentru funcţiile analitice

Una dintre cele mai importante noţiuni introduse de Amendamentul OLAP al SQL-99 este *fereastra. Fereastra* se defineşte în cadrul unei partiţii şi se referă la intervalul liniilor luat în calcul pentru *linia curentă.* Mărimea ferestrei se poate specifica fie *fizic,* printr-un număr de linii, fie *logic,* printr-un interval de tip dată calendaristică/timp sau interval de valori. Calculele se efectuează pentru fiecare linie din cadrul ferestrei, fereastră mişcătoare între o poziţie de start şi una de final. Linia curentă serveşte ca punct de referinţă pentru determinarea începutului şi sfârşitului ferestrei.

Specificaţiile unei ferestre privesc trei componente: partiţionarea, ordonarea şi grupurile de agregare. Orice funcţie de agregare poate fi utilizată în cadrul unei ferestre: `SUM, AVG, MIN, MAX, STDDEV, VARIANCE, COUNT.` Pe lângă acestea, Oracle 8i2 mai oferă suport pentru alte câteva funcţii statistice: `VAR_SAMP, VAR_POP, STDDEV_SAMP` ş.a.m.d., pe care, din lipsă de cunoştinţe de specialitate, nu le discutăm aici. Valorile agregate pot fi cumulative, mişcătoare sau centrate.

Să se afişeze, pentru fiecare zi, valoarea cumulată a vânzărilor după fiecare factură.

- o partiţie are dimensiunea unei zile; numărul de linii care o alcătuiesc depinde de numărul de facturi emise prezente în ziua respectivă;
- ordonarea în cadrul partiţiei se face după valoarea `Nr Fact;` prin urmare, poziţia de început a fiecărei partiţii este dată de prima factură din ziua respectivă, iar cea de sfârşit de factura ce are cel mare număr pentru `DataFact` ce defineşte partiţia curentă;
- fereastra pentru care se efectuează calculul, pentru fiecare linie, are o margine fixă - începutul partiţiei şi una mobilă, care este chiar linia curentă.

In SQL-99 apare o clauză nouă denumită chiar `WINDOW,` în care se declară cele trei elemente enumerate anterior, partiţionarea, ordonarea şi grupurile de agregare. Soluţia problemei formulate mai sus este:

```
SELECT DataFact AS Zi, NrFact, ValoareFact,
   SUM(ValoareFact) OVER W1 AS ValjCumulata FROM
   (SELECT DataFact, F.NrFact, SUM(Cantitate * PretUnit
        *  (1+ProcTVA)) AS ValoareFact FROM PRODUSE P,
   LINIIFACT LF, FACTURI F WHERE P.CodPr=LF.Codpr AND
   LF.NrFact=F.NrFact GROUP BY DataFact, F.NrFact )
   VALFACT WINDOW W1 AS
        (PARTITION BY DataFact ORDER
        BY NrFact ROWS UNBOUNDED
        PRECEDING)
```

Fereastra definită are numele `W1` şi se constituie în cadrul partiţiilor formate de valorile comune ale atributului `DataFact.` în cadrul partiţiilor, liniile sunt ordonate crescător după valorile atributului `NrFact,` iar intervalul liniilor asupra căruia se aplică funcţia de agregare `SUM (...)` este alcătuit din prima linie a partiţiei şi linia curentă `(ROWS UNBOUNDED PRECEDING).` Altă variantă SQL-99 permite declararea „în linie" a ferestrei, variantă utilizată de Oracle 8i2 şi DB2 v7, parametrii ferestrei fiind definiţi direct în clauza `SELECT,` după `OVER:`

```
SELECT DataFact AS Zi, NrFact, ValoareFact,
   SUM(ValoareFact) OVER
      (PARTITION BY DataFact ORDER BY NrFact ROWS
        UNBOUNDED PRECEDING)
   AS Val Cumulata
FROM
(SELECT DataFact, F.NrFact,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValoareFact FROM
   PRODUSE P, LINIIFACT LF, FACTURI F WHERE P.CodPr=LF.Codpr AND
   LF.NrFact=F.NrFact GROUP BY DataFact, F.NrFact ) VALFACT
```

ZI	NRFACT	VALOAREFACT	VAL CUMULATA
2000-08-01	1111	5399625J30[1]	5399625.00
2000-08-01	1112	1337560.00	6737185.00
2000-08-01	1113	1160250.00	7897435.00
2000-08-01	1114	6786570.00	14684005.00
2000-08-02	1115	i 6511 25.00	1 651 125.00
2000-08-02	1116	1383375.00	3034500.00
2000-08-03	1117	2320500.00	2320500.00
2000-08-04	1118	2052750.00	2052750.00
2000-08-07	1119	7242935.00	7242935.00
2000-08-07	1120	1066240.00	8309175.00
2000-08-07	1121	5438300.00	^ 13747475.00

Figura 7.32. Valoarea cumulată, după fiecare factură, pentru fiecare zi

Interogarea următoare, al cărei rezultat este cel din figura 7.33, ilustrează patru moduri de construire a unei ferestre:

Coloana `Cumul Prec` calculează suma valorilor facturilor din ziua respectivă, începând cu prima şi terminând cu factura curentă `(ROWS UNBOUNDED PRECEDING);`

Coloana `Crt_Precl` însumează valoarea facturii curente şi cea a precedentei `(ROWS 1` PRECEDING*);

La calculul `Precl Act Urmaţi` se însumează valorile facturilor: curentă, precedentă şi următoare `(ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING);` în fine, `Cumul_Urmat` însumează valoarea facturii curente şi tuturor celor ce-i succed în ziua respectivă `(ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING).`

```
SELECT DataFact AS Zi, NrFact, ValoareFact AS Val_Fact,
   SUM(ValoareFact) OVER (PARTITION BY DataFact ORDER BY NrFact
      ROWS UNBOUNDED PRECEDING)
      AS Cumul_Prec,
   SUM(ValoareFact) OVER (PARTITION BY DataFact ORDER BY NrFact
      ROWS 1 PRECEDING)
      AS Crt_Precl,
   SUM(ValoareFact) OVER (PARTITION BY DataFact ORDER BY NrFact
      ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
      AS Precl_Act_Urmatl,
   SUM(ValoareFact) OVER
        (PARTITION BY DataFact ORDER BY
        NrFact ROWS BETWEEN CURRENT ROW
        AND CJNBOUNDED FOLLOWING)
       AS Cumul_Urmat FROM
   (SELECT DataFact, F.NrFact,
        SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValoareFact FROM
       PRODUSE P, LINIIFACT LF, FACTURI F WHERE P.CodPr=LF.Codpr AND
       LF.NrFact=F.NrFact GROUP BY DataFact, F.NrFact ) VALFACT
```

21	NRFACT	VAL_FACT	CUMUL_PREC	CRT.PREC1	PREC1_ACT_URMAT1	CUMUL_URMAT
2000-08-01	___ 1111	.... 5399625.00	..... 5399625.00	.... 5399625.00	....6737185.00	_____ 14684005.00
2000-08-01"	1112	1337560.00	6737185.00	67371 85.00	7897435.00	928438 o'00
2000-08-01	1113	1160250.00	7897435.00	2497810.00	9284380.00	7946820.00
2000-08-01	1114	6786570.00	1 4684005.00	7946820.00	7946820.00	6786570.00
2000-08-02	1115	1651125.00	1651125.00	1 651 1 25.00	3034500.00	3034500 00
2000-08-02	1116	1383375 00	3034500.00	3034500.00	3034500.00	1383375.00
2000-08-03	1117	2320500.00	2320500.00	2320500.00	2320500.00	2320500.00
2000-08-04	1118	2052750.00	2052750.00	2052750.00	2052750.00	2052750.00
2000-08-07	1119	7242935.00	7242935.00	7242935.00	8309175.00	13747475.00
2000-08-07	1120	1 066240.00	8309175.00	83091 75.00	13747475.00	6504540.00
2000-08-07	1121	5438300.00	1 3747475.00	6504540.00	6504540.00	5438300.00

Figura 7.33. Patru moduri de definire a ferestrelor

Dacă în SELECT-ul precedent mărimea ferestrelor era definită fizic, prin numărul de linii ce precedă sau succedă liniei curente, în următoarea interogare se pun faţă în faţă ferestre definite fizic, prin număr de linii, şi logic, prin intervale (de zile).

```
SELECT Client, DataFact AS Zi, Vinzari,
    SUM(Vinzari) OVER
      (PARTITION BY Client ORDER BY DataFact ROWS 1 PRECEDING)
      AS Crt_Precl_Linii,
```

```
SUM(Vinzari) OVER
  (PARTITION BY Client ORDER BY DataFact RANGE INTERVAL '1'
  DAY PRECEDING)
AS Crt_Precl_Zile,
SUM(Vinzari) OVER
             (PARTITION BY Client ORDER BY DataFact
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
AS Pl_Act_Ul_Linii,
SUM(Vinzari) OVER
             (PARTITION BY Client ORDER BY DataFact
RANGE BETWEEN INTERVAL '1' DAY PRECEDING AND
```

```
    INTERVAL * 1' DAY FOLLOWING)
    AS Pl_Act_Ul_Zile
FROM
    (SELECT DenCl AS Client, DataFact,
    SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
    PRODUSE P, LINIIFACT LF, FACTURI F, CLIENŢI C WHERE
    P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND
    F.CodCl=C.CodCl GROUP BY DenCl, DataFact ) VALFACT
```

Coloana `Crt Precl Linii` însumează, după cum am văzut în precedenta interogare, valoarea facturii curente şi cea a precedentei `(ROWS 1 PRECEDING);`

La calculul valorii `Crt_Precl_Linii` se ia în calcul linia precedentă, numai dacă data căreia îi este asociată este imediat înaintea datei din linia curentă `(RANGE INTERVAL '1' DAY PRECEDING)` - vezi, pentru *Client 1,* situaţia zilei de 7 august;

CLIENT	ZI	VINZARI	CRT_PREC1 JJMi	CRT_PREC1_ZILE	P1_ACT_U1_UNII	P1_ACT_U1_ZILE
Client 1 SRL	2000-08-01	5399625	5399625	5399625	7050750	7050750
Client 1 SRL	2000-08-02	1651125	7050750	7050750	9371250	9371250
Client 1 SRL	2000-08-03	2320500	3971625	3971625	6024375	6024375
Client 1 SRL	2000-08-04	2052750	4373250	4373250	5439490	4373250
Client 1 SRL	2000-08-07	1066240	3118990	1066240	3118990	1066240
Client 2 SA	2000-08-01	1160250	1160250	1160250	1160250	1160250
Client 3 SRL	2000-08-07	7242935	7242935	7242935	7242935	7242935
Client 4	2000-08-07	5438300	5438300	5438300	5438300	5438300
Client S SRL	2000-08-01	1337560	1337560	1337560	1337560	1337560
Client 6 SA	2000-08-01	6786570	6786570	6786570	j 6786570	6786570
Client 7 SRL	2000-08-02	1383375	1383375	1383375	1383375	1383375

Figura 7.34. Ferestrele definite logic şi fizic

Curios, deşi interogarea respectă specificaţiile SQL-99 şi funcţionează în Oracle 8i2, în DB2 v7 nu am reuşit să o facem operaţională.

7.5. Comparaţii şi ponderi în Oracle 8i2

Atunci când, într-o partiţie, se doreşte raportarea valorilor liniei curente la cele din prima sau ultima linie, este rezonabil să se folosească funcţiile `FIRST_VALUE` şi `LAST_VALUE`.

Care este raportul zilnic al vânzărilor, relativ la prima zi de vânzări?

```
SELECT
   TO_CHAR(DataFact,'YYYY-MM-DD') AS Zi,
   TO_CHAR(Vinzari,'999999999999') AS Vinzari,
FIRST_VALUE(Vinzari) OVER (ORDER BY DataFact) AS Vnz_Zil,
ROUND(Vinzari / FIRST_VALUE(Vinzari) OVER
   (ORDER BY DataFact),2) AS Raport Crt Prima Zi FROM    '
         "                              ~
(SELECT DataFact, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS
Vinzari FROM PRODUSE P, LINIIFACT LF, FACTURI F WHERE
```

```
   P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact GROUP BY DataFact ) -
```

...	VINZARI	VNZ_ZI1	RAPORT_CRT_PRIMA_ZI
2000-08-01	14684005	14684005	1
2000-08-02	3034500	14684005	.21
2000-08-03	2320500	14684005	.16
2000-08-04	2052750	14684005	.14
2000-08-07	13747475	14684005	.94

Figura 7.35. Exemplu de utilizare a funcţiei FIRST_VALUE

Să se determine ziua în care s-a vândut cel mai bine fiecare produs.

Una dintre soluţii se bazează pe o subconsultare ce întrebuinţează funcţia FIRST VALUE:

```
SELECT DenPr, TO_CHAR(DataFact,     'YYYY-MM-DD') AS DataMax,
      TO_CHAR(Vnz_Max,'999999999999') AS Vnz_Max
FROM
      (SELECT DenPr, DataFact,
           SUM(Cantitate * PretUnit * (1+ProcTVA))
           AS Vinzari,
      FIRST_VALUE (SUM (Cantitate * PretUni"" * (1 + ProcTVA)))
      OVER (PARTITION BY DenPr ORDER BY
      SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
      AS Vnz_Max FROM PRODUSE P, LINIIFACT LF, FACTURI F WHERE
   P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact GROUP BY DenPr, DataFact
   )

WHERE Vinzari = Vnz_Max
```

Subconsultarea:

```
SELECT DenPr, DataFact,
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari,
   FIRST_VALUE(SUM(Cantitate * PretUnit * (1+ProcTVA))) OVER
   (PARTITION BY DenPr
   ORDER BY SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC) AS
Vnz_Max FROM PRODUSE P, LINIIFACT LF, FACTURI F WHERE
P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact GROUP BY DenPr, DataFact
```

obţine rezultatul din figura 7.36.

DENPR	DATAFACT	VINZARI	VNZ_MAX
Produs 1	2000-08-04	1660050	1660050
Produs 1	2000-08-03	1130500	1680050
Produs 1	2000-08-01	595000	1660050
Produs 2	2000-08-01	3969245	3969245
Produs 2	2000-08-02	3034500	3969245
Produs 2	2000-08-07	2769725	3969245
Produs 2	2000-08-03	1190000	3969245
Produs 2	2000-08-04	392700	3969245
Produs 3	2000-08-01	357000	357000
Produs 3	2000-08-07	333200	357000
Produs 4	2000-08-07	833000	833000
Produs 4	2000-08-01	564060	833000
Produs 5	2000-08-07	9811550	9811550
Produs 5	2000-08-01	9198700	9811550

Figura 7.36. Determinarea, pentru fiecare produs, a vânzărilor şi a maximului zilnic în final,

răspunsul problemei este cel din figura 7.37.

DENPR	DATAMAX	VNZ_MAX
Produs 1	2000-08-04	1660050
Produs 2	2000-08-01	3969245
Produs 3	2000-08-01	357000
Produs 4	2000-08-07	833000
Produs 5	2000-08-07	9811550

Figura 7.37. Zilele în care s-a vândut cel mai bine fiecare produs

Pentru respectarea adevărului istoric, se cuvine de remarcat că, în locul „cosmopolitei" funcţii FIRST_VALUE, am fi putut folosi în subconsultare şi „tocita" MAX, bineînţeles, într-o haină nouă:

```
SELECT DenPr, DataFact, SUM(Cantitate * PretUnit
      *  (1+ProcTVA)) AS Vinzari,
   MAX(SUM(Cantitate * PretUnit * (1+ProcTVA)))
      OVER (PARTITION BY DenPr ORDER BY
      SUM(Cantitate * PretUnit * (1+ProcTVA)) DESC)
   AS Vnz_Max FROM PRODUSE P, LINIIFACT LF, FACTURI F WHERE
P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact GROUP BY DenPr, DataFact
```

Atunci când se doreşte determinarea: contribuţiei unui produs la totalul vânzărilor, părţii unui client în totalul datornicilor, procentului unui produs din valoarea unei facturi, ponderii salariului directorului general (sau al şefilor de compartimente) în totalul fondului de salarii al unei întreprinderi ş.a.m.d. se poate apela la serviciile funcţiei RATIO TO REPORT, care calculează raportul dintre o valoare (atribut/expresie) şi suma totală a valorii respective pentru toate liniile din fereastră.
Să se afişeze structura vânzărilor pe judeţe.

```
SELECT Judeţ, Vinzari, ROUND(RATIO_TO_REPORT(Vinzari)
    OVER ( ), 2 )  AS Pondere_Judet FROM
    (SELECT Judeţ, SUM(Cantitate * PretUnit * (1+ProcTVA))
          AS Vinzari FROM PRODUSE P, LINIIFACT LF,
    FACTURI F,
          CLIENŢI C, LOCALITATI L, JUDEŢE J WHERE
```

```
    P.CodPr=LF.Codpr AND LF.NrFact=F.NrFact AND F.CodCl=C.CodCI
          AND C.CodPost=L.CodPost AND L.Jud=J.Jud GROUP
    BY Judeţ )
```

După cum se observă, avantajul funcţiei este că nu necesită precizarea explicită a numitorului. Rezultatul interogării este prezentat în figura 7.38.

JUDEŢ	VINZARI	PONDERE_JUDET
Iasi	19088790	.53
Neamţ	6786570	.19
Timiş	2720935	.08
Vaslui	7242935	.2

Figura 7.38. Contribuţia fiecărui judeţ în totalul vânzărilor

Să se calculeze contribuţia produselor la valoarea fiecărei facturi.

```
SELECT Factura, Produs, TO_CHAR(Vinzari, '999999999999 ' ) AS Vinzari,
    ROUND(RATIO_TO_REPORT(Vinzari) OVER (PARTITION BY
       Factura),2)
       *  100 ||'%' AS Pondere_Prod_Fact
FROM
    (SELECT NrFact AS Factura, DenPr AS Produs,
       SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM
    PRODUSE P, LINIIFACT LF WHERE P.CodPr=LF.Codpr GROUP BY
    NrFact, DenPr
```

FACTURA	PRODUS	VINZARI	PONDERE_PROD_FACT
1111	Produs 1	595000	11%
11-11 «Produs 2		937125	17%
1111	Produs 5	3867500	72%
1112	Produs 2	980560	73%
1112	Produs 3	357000	27%
1113	Produs 2	1160250	100%
1114	Produs 2	891310	13%
1114	Produs 4	564060	8%
1114	Produs 5	5331200	79%
1115	Produs 2	1651125	100%
1118	Produs 2	1383375	100%
1117	Produs 1	1130500	49%
1117	Produs 2	1190000	51%
1118	Produs 1	1660050	81%
1118	Produs 2	392700	19%
1119	Produs 2	453985	6%
1119	Produs 3	333200	5%
1119	Produs 4	833000	12%
1119	Produs 5	5622750	78%
1120	Produs 2	1066240	100%
1121	Produs 2	1249500	23%
1121	Produs 5	4188800	77%

Figura 7.39, Ponderea fiecărui produs în factură

La analiza dinamicii temporale a unei mărimi se raportează valoarea din linia curentă la cea dintr-o altă linie plasată la o anumită distanţă. Spre exemplu, pentru compararea vânzărilor fiecărei luni calendaristice cu vânzările din aceeaşi lună din anul precedent, va trebui să împărţim valoarea de pe linia curentă la o valoare aflată cu 12 linii/luni înainte. Apare astfel noţiunea de *deplasament înapoi* (LAG) şi *înainte* (LEAD). Fireşte, soluţia funcţionează numai dacă nu există luni de întrerupere, adică dacă intervalul este continuu.

în interogarea următoare, al cărei rezultat este prezentat în figura 7.40, deplasamentul este, pentru ambele funcţii, 1,

```
SELECT TO_CHAR(Zi,'YYYY-MM-DD') AS Zi, Vinzari,
    LAG(Vinzari,1) OVER (ORDER BY Zi) AS Vnz_LAG,
    LEAD(Vinzari,1) OVER (ORDER BY Zi) AS Vnz_LEAD FROM
    (SELECT DataFact AS Zi,
        SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE
    P, LINIIFACT LF, FACTURI F WHERE P.CodPr=LF.CodPr AND
    LF.NrFact=F.NrFact GROUP BY DataFact)
```

Funcţiile LAG şi LEAD extrag vânzările din ziua precedentă, respectiv următoare. Locurile libere din cadrul listei reprezintă valori NULL.

ZI	VINZARI	VNZ_LAG	VNZ_LEAO
2000-08-01	14684005		3034500
2000-08-02	3034500	14684005	2320500
2000-08-03	2320500	3034500	2052750
2000-08-04	2052750	2320500	13747475
2000-08-07	13747475	2052750	

Figura 7.40. Vânzările din ziua curentă, precedentă şi următoare

Care este evoluția vânzărilor de la o zi la alta?

```
SELECT TO_CHAR(Zi,'YYYY-MM-DD') AS Zi,
    TO_CHAR(Vinzari, '999999999999') AS Vinzari,
    TO_CHAR(LAG(Vinzari, 1) OVER
        (ORDER BY Zi),'999999999999') AS Vnz_Preced, TO_CHAR(Vinzari -
    LAG(Vinzari,1) OVER "
    (ORDER BY Zi),'999999999999') AS Diferența FROM
    (SELECT DataFact AS Zi,
        SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari FROM PRODUSE P, LINIIFACT
    LF, FACTURI F WHERE P.CodPr=LF.CodPr AND LF.NrFact=F.NrFact GROUP BY DataFact )
```

Capitolul 8

OBIECTE DIN SCHEMA BAZEI DE DATE. EXTENSII PROCEDURALE ALE SQL

Schema unei baze de date cuprinde, la SGBD-urile moderne, o largă varietate de obiecte, pe lângă ceea ce rămâne esenţial - tabela. în SQL-89, singurele obiecte din schemă la care utilizatorii aveau acces erau tabela şi tabela virtuală *(view).* SQL-92 consacră o tipologie mult mai generoasă, din care o parte rămâne încă neimplementată în produsele comerciale. La noţiunile amintite în primul capitol - tabelă, tabelă virtuală, declanşator/ procedură stocată - mai putem adăuga: tabelă temporară, aserţiune, domeniu, secvenţă, legătură cu altă bază de date/instanţă etc. Acest ultim capitol este dedicat aprofundării unora dintre noţiunile deja discutate, prezentării unor noi obiecte, precum şi modului în care pot fi obţinute informaţii despre diferite componente din schema bazei.

8.1. Dicţionarul bazei de date. Reguli de validare avansate

In primul capitol, preponderent terminologic, prezentam cele două modalităţi de abordare a unei baze de date, unul care are în vedere aspectul constant, structura (intensia), şi celălalt preocupat de conţinut efectiv (extensia). De asemenea, am discutat despre rolul dicţionarului de date, cel de stocare a schemei bazei, dicţionar numit şi *container* sau *catalog-sistem.* Dincolo de nuanţele specifice fiecărui SGBD, schema unei baze de date poate fi privită ca o descriere a bazei, descriere generată prin intermediul limbajului de definire a datelor (DDL) propriu SGBD-ului respectiv. Cele mai întrebuinţate obiecte ale bazei sunt: tabelele, tabelele virtuale, domeniile, restricţiile, privilegiile, procedurile stocate, sinonimele, indecşii, clusterele etc.

Sihtaxa SQL-92 pentru crearea schemei unei baze de date este relativ simplă:

CREATE SCHEMA <clauză de denumire>
    [<specificarea setului de caractere>]
    [<element>...]

unde:

<clauză de denumire> : : = =
<nume schemă>
I AUTHORIZATION <identificator de autorizare> I <nume schemă>
AUTHORIZATION
< identificator de autorizare >
< specificarea setului de caractere > : : = =
DEFAULT CHARACTER SET
<specificatorul setului de caractere>
< element > : : = =
< definiţie domeniu >
I < definiţie tabelă >
I < definiţie tabelă virtuală >
I < comandă pentru acordare de drepturi >
I < definiţie aserţiune >

Atât obiectele, cât mai ales dicţionarul bazei, prezintă diferenţe semnificative de la SGBD la SGBD. Dacă e să începem cu una dintre cele mai simple forme de dicţionar de date, putem face referire la containerul *(database container)* Visual FoxPro. Acesta este o tabelă specială cu structura prezentată în figura 8.1.

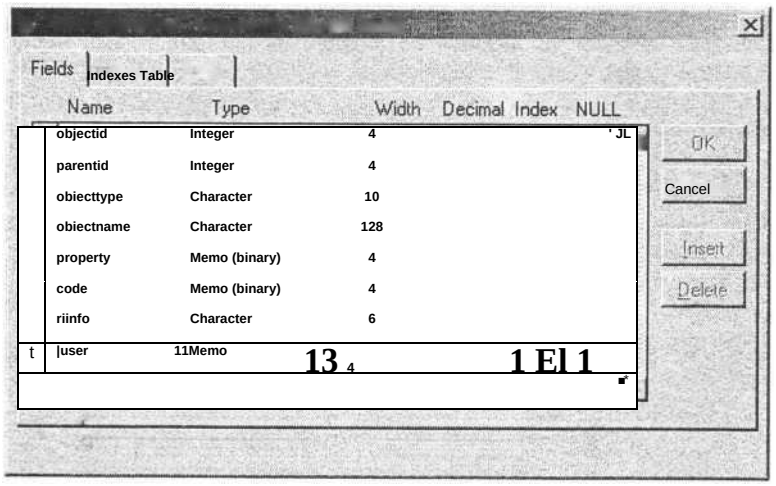


Figura 8.1. Structura containerului unei 8D în VFP

Fiecare obiect din bază este identificat printr-un număr întreg unic - objectid. Ierarhia obiectelor (de exemplu, atributele unei tabele) este reflectată prin câmpul parentid. Tipologia obiectelor din container (atributul obj ecttype) este mai săracă în VFP: Database, Table, View, Field, Index şi Relation. Dacă primele nu necesită comentarii, despre Relation trebuie spus că reflectă o legătură permanentă între două tabele, legătură ce nu asigură automat şi restricţiile referenţiale. Procedurile stocate nu sunt un tip special, ci au trei obiecte corespondente, StoredProceduresSource, StoredProceduresObject şi StoredProceduresDependencies, toate de tipul Database.

în Oracle, lucrurile sunt cu mult mai analitice. Dicţionarul este un set de tabele şi tabele derivate accesibile diferenţiat, în funcţie de rolul utilizatorului. Dacă numele acestora este prefixat de USER, atunci este vorba despre obiecte ce aparţin utilizatorului curent; ALL desemnează, în plus, şi obiectele la care utilizatorul curent are acces, dar proprietarul este un alt utilizator, iar prefixul DBA (*Data Base Administrator)* se referă la toate obiectele bazei, indiferent de „proprietar".

Tabela principală a catalogului este numită DICTIONARY şi are doar două coloane, TABLE_NAME şi COMMENTS. Aflarea tuturor tabelelor virtuale ce oferă informaţii despre obiectele din schemă este posibilă prin interogarea:

```
SELECT *
FROM DICTIONARY
```

Pentru non administratori, inventarierea obiectelor din schema proprie (tabele, tabele virtuale, sinonime, proceduri, funcţii, pachete, secvenţe etc.) este posibilă folosind o altă tabelă din dicţionarul datelor - USER_OBJECTS:

```
SELECT *
FROM USER_OBJECTS
```

DB2 (versiunea 7) administrează două seturi de tabele derivate în cadrul dicţionarului de date, în schemele SYSCAT şi SYSSTAT. Ambele seturi sunt accesibile utilizatorilor. Tabelele virtuale din SYSSTAT reprezintă un subset al celor din SYSCAT, singurul avantaj fiind că o parte din atribute sunt actualizabile.

Despre crearea tabelelor am discutat pe îndelete în capitolul 3. Am trecut în revistă principalele clauze pentru declararea restricţiilor: NOT NULL, PRIMARY KEY, UNIQUE, CHECK, FOREIGN KEY.

Ceea nu am prezentat în capitolul respectiv ţine de crearea unei tabele pe baza unei alte tabele, prin preluarea unuia sau mai multor atribute, eventual chiar definind noi atribute. Spre exemplu, dacă în DB2 se doreşte ca din tabela FACTURI să se arhiveze toate facturile emise în 1999 într-o altă tabelă, denumită FACTURI 1999, se poate folosi următoarea secvenţă de comenzi SQL:

```
CREATE TABLE FACTURI_1999 AS
    (SELECT *
    FROM FACTURI)
    DEFINITION ONLY ;
INSERT INTO FACTURI_1999 (SELECT *
    FROM FACTURI
    WHERE DataFact BETWEEN '1999-01-01' AND '1999-12-31'
    ) ;
COMMIT ;
```

Lucrurile nu diferă prea mult în Oracle, dar crearea tabelei FACTURI_1999 şi popularea sa pot fi realizate dintr-o singură mişcare:

```
CREATE TABLE FACTURI_1999 AS
    (SELECT *
    FROM FACTURI WHERE DataFact BETWEEN
TO_DATE('1999-01-01','YYYY-MM-DD') AND TO
DATE('1999-12-31','YYYY-MM-DD'))
```

în Visual FoxPro, crearea unei tabele printr-o combinaţie CREATE TABLE / SELECT nu este posibilă. Prin clauza FROM ARRAY, o tabelă poate prelua structura şi conţinutul unui masiv (ARRAY). Revenind la problema noastră, rezolvarea este la fel de simplă, dar folosind clauza INTO:

```
SELECT * ;
FROM FACTURI ;
INTO TABLE FACTURI_1999 ;
WHERE DataFact BETWEEN {"1999-01-01} AND {^1999-12-31 }
```

Tot în capitolul 3 prezentam clauza CHECK pentru declararea unor restricţii la nivel de atribut. Din păcate, nici unul dintre dialectele SQL ale celor trei produse - DB2, Oracle şi VFP - nu suportă clauza SQL-92 legată de folosirea subconsultărilor. Spre exemplu, dacă dorim ca în tabela LINIIFACT să instituim restricţia ca, în urma oricărei modificări a acestei tabele, să nu existe nici o factură fără măcar o linie, în SQL-92 se poate specifica:

```
ALTER TABLE liniifact ADD CONSTRAINT macar_o_linie CHECK (EXISTS
(SELECT 1 FROM facturi
WHERE facturi.NrFact=liniifact.NrFact))
```

Nici în privinţa domeniilor şi aserţiunilor implementările SQL ale produselor comerciale nu stau grozav, aşa că în continuare ne ocupăm de tabele virtuale, de proceduri stocate şi declanşatoare.

în VFP, dacă se doreşte extragerea tuturor tabelelor din bază, se foloseşte o consultare relativ simplă:

```
SELECT * FROM vinzari.dbc WHERE OBJECTTYPE=1 Table'
```

Răspunsul este cel din figura 8.2. Atributul Property furnizează, prin altele (numele fişerului .DBF de pe disc, indexul primar al tabelei), şi numele eventualelor declanşatoare ale tabelei.

	Objectid	Parenlid	Obiecttype	Objectname	Property	Code	Riinfo	Useil
	ei	1!	Table	judeţe	Memo	memo;	memo!	
	121	1	Table	localitati	Memo	memo	memo!	
	19!	1!	Table	clienţi	Memo	memo	memo!	
	291	11	Table	persoane	Memo	memo	memo!	
	43i	1!	Table	persclienti	Memo	memo	; memo!	
	52!	1!	Table	produse	Memo	memo	memo!	
	59!	11	Table	facturi	Memo	memo	memo!	
	63	1!	Table	liniifact	Memo	memo	memo!	
	74!	1!	Table	incasari	Memo	memo!	memoi	
	81!	1	Table	incasfact	Memo	memo;	: memoi ,	

Figura 8.2. Extragerea din containerul BD în VFP a informaţiilor despre tabele

Informaţii despre atributele tabelelor pot fi obţinute în mod asemănător:

```
SELECT v2.objectname, vl.objectid, vl.objectname, ; left(vl.property, 250) ;
```

```
FROM vinzari.dbc vi INNER JOIN vinzari.dbc v2 ;
    ON vl.parentid=v2.objectid ;
WHERE v2.OBJECTTYPE='Table' AND vl.OBJECTTYPE='Field'                    ;
ORDER BY VI.objectid
```

Din păcate, modul de prezentare a valorilor implicite, regulilor de validare şi mesajelor de eroare afişate la încălcarea regulilor de validare la nivel de câmp este destul de criptic şi dificil de folosit în aplicaţii.

Dată fiind structura mult mai analitică a dicţionarului bazei (catalogul-sistem), în Oracle sunt necesare mai multe interogări:

- pentru aflarea tabelelor din schema curentă:

```
SELECT TABLE_NAME
FROM USER_TABLES
```

- pentru aflarea tuturor tabelelor, tabelelor virtuale, sinonimelor şi secvenţelor din schema utilizatorului curent:

```
SELECT TABLE_NAME, TABLE_TYPE
FROM USER_CATALOG
```

- pentru afişarea informaţiilor despre atributele unei tabele:

```
SELECT *
FROM USER__TAB_COLUMNS
WHERE TABLE_NAME='FACTURI [1]
```

- pentru listarea informaţiilor despre tabelele derivate:

```
SELECT *
FROM USER_VIEWS
```

- afişarea restricţiilor din schema curentă:

```
SELECT *
FROM USER_CONSTRAINTS
```

Lucrurile stau oarecum asemănător şi în DB2:

- tabelele din schema FOTACHEM:

```
SELECT *
FROM SYSCAT.TABLES WHERE
TABSCHEMA='FOTACHEM'
```

- tabelele virtuale:

```
SELECT *
'FROM SYSCAT.VIEWS WHERE
VIEWSCHEMA='FOTACHEM'
```

- informaţii despre atributele unei tabele:

```
SELECT *
FROM SYSCAT.COLUMNS
WHERE TABNAME='FACTURI'
```

- restricţiile de tip CHECK din tabela FACTURI:

```
SELECT *
FROM SYSCAT.COLUMNS
WHERE TABNAME='FACTURI'
```

- coloanele tabelei FACTURI pentru care au fost declarate restricţii de tip CHECK:

```
SELECT *
FROM SYSCAT.COLCHECKS
WHERE TABNAME='FACTURI'                    .
```

8.2. Tabele virtuale în Oracle şi VFP

în mod obişnuit, tabelele derivate sunt percepute ca tabele noi, construite prin subconsultări aplicate asupra tabelelor de bază sau altor *view*-uri. Standardul SQL-92 defineşte *view*-urile ca tabele virtuale ce sunt materializate atunci când este invocat numele lor. Materializarea înseamnă, de fapt, execuţia frazei SELECT ce constituie definiţia tabelei virtuale şi popularea cu înregistrări care sunt, de fapt, rezultatul interogării. Formatul general al comenzii de creare a unei asemenea tabele virtuale este:

CREATE VIEW <nume-tabelă-virtuală> [<listă-coloane>] AS <frază SELECT>
[WITH [<clauză de nivele>] CHECK OPTION]

unde
<clauză de nivele> ::= CASCADED | LOCAL

O dată creată tabela virtuală, definiţia sa este salvată în schema bazei. Ulterior, ori de câte ori este necesar, la deschiderea/reîmprospătarea tabelei derivate, aceasta este (re)populată cu înregistrări extrase din cele ale tabelelor propriu-zise ce apar în clauza FROM a interogării. O tabelă virtuală poate fi deci privită şi ca o expresie de subconsultare a unei tabele persistente, stocată în bază şi invocată prin numele său.

Potrivit SQL-92, unei tabele virtuale nu i se pot asocia indecşi şi nici defini restricţii, deşi unele SGBD-uri optimizează lucrul cu *view*-urile folosind indecşii tabelelor persistente. Numele său este unic şi nu se poate autoreferi, deşi un *view* poate fi creat pe baza unei combinaţii de tabele persistente şi/sau alte *view*-uri.

SQL-92 introduce şi noţiunea de *tabelă derivată,* definită asemănător celei virtuale, dar pentru care se pot defini restricţii şi asocia indecşi, fiind, spre deosebire de tabela virtuală, persistentă[58].

<subconsultare-tabelă> AS <nume-tabelă-virtuală> [<listă-coloane>]

în ceea ce ne priveşte, acest paragraf este dedicat tabelelor virtuale, problematica expusă fiind o continuare a ceea ce a fost deja prezentat în paragraful 1.5. După cum am văzut

atunci, cele mai dificile probleme ale *view*-urilor ţin de actualizarea lor, de fapt, de propagarea actualizărilor tabelelor virtuale în tabelele de bază din care sunt construite.

La creare, o tabelă virtuală poate fi declarată actualizabilă sau non-actualizabilă *(Read-Only).* Pentru a fi actualizabilă, o tabelă virtuală trebuie să aibă fiecare linie a sa asociată unei singure linii din tabelele bazei. La modificarea unei linii din *view,* propagarea poate fi făcută, în aceste condiţii, fără probleme de ambiguitate.

58. Preluare din [Celko2000], p. 266.

Firește, o tabelă derivată de genul:

```
CREATE VIEW vJudetel AS
    SELECT *
    FROM Județe
    WHERE regiune = 'Moldova' OR regiune = 'Dobrogea'
```

nu va crea probleme insolubile la actualizare. în schimb, deşi cu un conţinut identic vJudetel, tabela virtuală vJudete2, creată prin comanda care urmează, nu este actualizabilă nici în SQL-92, nici în majoritatea SGBD-urilor importante.

```
CREATE VIEW vJudete2 AS
    SELECT *
    FROM Județe
    WHERE regiune = 'Moldova'
    UNION
    SELECT *
        FROM Județe
        WHERE regiune = 'Dobrogea'
```

Regulile actualizării tabelelor în SQL-92 sunt foarte stricte[59]:

1. Tabela virtuală trebuie derivată dintr-un SELECT cu o singură tabelă de bază; mai multe tabele de bază înseamnă mai multe niveluri ale tabelelor virtuale. Spre exemplu, pentru a construi o tabelă virtuală vCLIENTI în care pot fi modificate: denumirea clientului, adresa sa, denumirea localităţii în care îşi are sediul şi numele judeţului, sunt necesare următoarele *view*-uri:

```
CREATE VIEW vJudete AS
    SELECT *
    FROM Județe
CREATE VIEW vLocalitati AS
    SELECT CodPost, Loc, vJudete.Jud, Judeţ, Regiune FROM LOCALITATI
    INNER JOIN vJUDETE ON LOCALITATI.Jud = vJUDETE.Jud
CREATE VIEW vClienti AS
    SELECT CodCI, DenCl, Adresa, vLocalitati.CodPost,
        Loc, Jud, Judeţ, Regiune FROM CLIENŢI INNER JOIN
    VLOCALITATI
        ON CLIENŢI.CodPost = VLOCALITATI.CodPost
```

59. [Celko99], p. 57.

2. Tabelele virtuale trebuie să includă toate coloanele cheilor primare/alternative ale tabelelor de bază. O linie dintr-o tabelă virtuală trebuiesă corespundă unei singure linii din tabela de bază.
3. Toate coloanele neincluse în *view* trebuie să permită valori NULL sau să prezintevalori implicite. Altminteri inserarea unei linii într-o tabelă derivată nu este posibilă

Nu poate fi actualizată o tabelă virtuală creată prin fraze SELECT în care apar:

- clauze GROUP BY / HAVING,
- funcţii-agregat,
- coloane calculate,
- UNION,
- » INTESECT,
- EXCEPT (MINUS),
- SELECT DISTINCT.

Din punctul de vedere al actualizării tabelelor virtuale, SQL-92 este mult mai limitat decât o cer regulile lui Codd, preferându-se siguranţa. Multe SGBD-uri sunt însă mai îngăduitoare în această privinţă. Mai mult, în SQL-3 şi o serie de SGBD-uri, prin mecanisme de genul declanşatoarelor INSTEAD OF, multe probleme ale tabelelor virtuale, insolubile în SQL-92, îşi găsesc rezolvarea.

Atunci când tabela virtuală este actualizabilă, se poate folosi clauza WITH CHECK OPTION pentru un mai bun control al modificărilor. vJudeteMoldova conţine înregistrările tabelei JUDEŢE pentru care valoarea atributului Regiune este *Moldova-, '*

```
CREATE VIEW vJudeteMoldova AS SELECT *
    FROM judeţe
    WHERE Regiune='Moldova'
    WITH CHECK OPTION
```

Comanda INSERT în formatul:

INSERT INTO vJudeteMoldova VALUES . ('BC', ’Bacau', 'Moldova') se execută, fără probleme, ceea ce nu este valabil şi pentru:

```
INSERT INTO vJudeteMoldova VALUES ('AR', 'Arad', 'Banat')
```

care va genera un mesaj de eroare din cauza regiunii ce nu este acceptată în tabela virtuală, graţie clauzei CHECK OPTION. în SQL-92, clauza de verificare este, implicit, declanşată în cascadă (CASCADED), adică atât pentru tabela virtuală curentă, cât şi pentru cele pe baza cărora a fost construită, nivel cu nivel. Cu toate acestea, CASCADED se poate înlocui cu LOCAL, caz în care se verifică numai WHERE-ul prezentei tabele virtuale, fară a se testa modul în care modificările afectează tabelele derivate de pe nivelurile inferioare.

Ştergerea unei tabele virtuale din schema bazei de date se realizează prin comanda DROP VIEW. -

DROP VIEW cnume-tabelă-virtuală> <comportament> unde

<comportainent> ::= CASCADE | RESTRICT

Prin clauza `CASCADE` se şterg atât tabela virtuală curentă, cât şi toate cele create pe baza acesteia, în timp ce `RESTRICT` interzice operaţiunea atâta timp cât există măcar un *view* construit pe baza tabelei virtuale curente.

Tabele virtuale în Oracle

în Oracle, aflarea atât a frazei `SELECT` de creare, cât şi a modului în care o tabelă derivată poate fi actualizată presupune consultarea dicţionarului de date. Spre exemplu, interogarea prin care a fost creată `vJudeteMoldova` se vizualizează prin:

```
SELECT TEXT FROM
USER_VIEWS
WHERE VIEW_NAME = 'VJUDETEMOLDOVA'
```

iar

interogarea:

```
SELECT column_name, updatable, insertable, deletable
FROM user_updatable_columns
WHERE table_name = 'VJUDETEMOLDOVA'
```

va determina afişarea rezultatului din figura 8.3.

COLUMN_NAME | UPDATABLE	INSERTABLE	DELETABLE
JUD YES	YES	YES
JUDEŢ YES	YES	YES
REGIUNE YES	YES	YES

Figura 8.3. Posibilităţi de actualizare în tabela derivată `vJudeteMoldova`

Rezultă că asupra acestei tabele derivate poate fi operată toată gama de operaţiuni pentru toate atributele. în schimb, dacă pe baza acesteia creăm o alta pentru vizualizarea şi actualizarea localităţilor din judeţul Iaşi:

```
CREATE VIEW vLocJudlasi AS
   SELECT localitati.codpost, localitati.loc,
   localitati.jud, judeţ, regiune FROM localitati,
   vJudeteMoldova WHERE localitati.jud =
   vJudeteMoldova.jud AND localitati.jud='IS'
WITH CHECK OPTION
```

situaţia se prezintă diferit - vezi figura 8.4.

COLUMN_NAME	UPDATABLE	INSERTABLE	DELETABLE
CODPOST	YES	YES	YES
LOC	YES	YES	YES
JUD	NO	NO	NO
JUDEŢ	NO	NO	NO
REGIUNE	NO	NO	NO

Prin urmare, orice tentativă de a modifica ultimele trei atribute va eşua. Ba mai mult, nici inserarea nu funcţionează. Comanda:

```
INSERT INTO vLocJudlasi VALUES ('68 68', 'Tg. Frumos', 'IS',
   'Iasi', 'Moldova')
```

va genera mesajul de eroare: `ORA-01733: virtual column not allowed here.` Chiar dacă schimbăm definiţia în:

```
CREATE OR REPLACE VIEW vLocJudlasi AS
   SELECT localitati.codpost, localitati.loc, localitati.jud, judeţ,
regiune FROM localitati, judeţe WHERE localitati.jud = judeţe.jud AND
localitati.jud='IS' WITH CHECK OPTION
```

rezultatul va fi identic[31]. Prin urmare, în `vLodasi` este posibilă numai modificarea atributelor `CodPost` şi `Loc`. Aceste probleme majore de actualizare în Oracle 8i pot fi rezolvate cu ajutorul declanşatoarelor, după cum vom vedea peste câteva pagini.

Tabele virtuale în Visual FoxPro

Visual FoxPro este un produs generos în ceea ce priveşte tabelele derivate. Acestea pot fi create atât grafic, cât şi prin program. Important este că pot fi definite explicit atributele care pot fi actualizate sau nu, modul de actualizare a tabelelor, ba chiar, mai mult, se pot defini indecşi şi reguli de validare chiar la nivel de *view*[32]. Programul prezentat în listingul 8.1 este grăitor în acest sens.

31 Pentru detalii privind actualizarea tabelelor virtuale în Oracle 8i, vezi şi [BobrowskiOl].
32 Vezi şi [Fotache&Strîmbei99-2],

Figura 8.4. Posibilităţi de actualizare în tabela derivată vLocJudlasi

Listing 8.1. Program Visual FoxPro 6 de creare a unei tabele derivate

```
*** CREAREA TABELEI DERIVATE vFacturi

IINCLUDE foxpro.h
IF IDBUSED('vinzari')
   OPEN DATABASE vinzari ENDIF
SET DATABASE TO vinzari

IF USED('vFacturi')
   SELECT vFacturi USE ENDIF

*  Fraza SQL de creare a tabelei virtuale
CREATE SQL VIEW vFacturi AS ;
   SELECT LF.NrFact, F.DataFact, Linie, LF.CodPr, DenPr, ;
      UM, Cantitate, PretUnit, Cantitate * PretUnit AS ;
      ValFaraTVA, Cantitate * PretUnit * ProcTVA AS TVA, ;
      Cantitate * PretUnit * (1 + ProcTVA) AS ValTotala ;
   FROM LINIIFACT LF INNER JOIN FACTURI F ON LF.NrFact=F.NrFact ;
      INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
   ORDER BY LF.NrFact, Linie
```

```
* Pe baza tabelei derivate, se va actualiza numai LINIIFACT
DBSETPROP('vFacturi', 'View', 'Tables', 'LINIIFACT')
* Se declara atributele cheie (primara)
DBSETPROP('vFacturi.NrFact', 'Field', 'KeyField', .t.)
DBSETPROP('vFacturi.Linie', 'Field', 'KeyField', .t.)
* Se declara ca actualizabile atributele cheie primara
DBSETPROP('vFacturi.NrFact', 'Field', 'Updatable', .t.)
DBSETPROP('vFacturi.Linie', 'Field', 'Updatable', .t.)
* Tipul modificărilor (UPDATE, nu INSERT-DELETE)
DBSETPROP('vFacturi', 'View', 'UpdateType', DB_UPDATE)
* La propagarea actualizarii in tabelele de baza, se verifica
* daca au intervenit modificări in continutul atributelor cheie
* si a celor actualizabile
DBSETPROP('vFacturi', 'View', 'WhereType', DB_KEYANDUPDATABLE)
* Semnalul final pentru propagarea modificărilor din
* tabela derivata in cea de baza
DBSETPROP('vFacturi', 'View', 'SendUpdates', .t.)
RETURN
```

O dată creată, tabela derivată trebuie deschisă explicit prin comanda USE, operaţiune în urma căreia are loc execuţia frazei SELECT şi popularea cu înregistrări. De remarcat că împrospătarea tabelei derivate cu cele mai noi înregistrări ale tabelelor de bază nu necesită închiderea şi deschiderea sa, funcţia REQUERY () asigurând acest lucru.

Afişarea tabelelor derivate din baza de date se obţine asemănător tabelelor „normale”:

```
SELECT * FROM vinzari.dbc WHERE OBJECTTYPE='View'
```

Atributul Property conţine fraza SELECT pe baza căreia a fost construită tabela virtuală. Cât despre obţinerea datelor despre atributele tabelelor derivate, iată interogarea:

```
SELECT v2.objectname, vl.objectid, vl.objectname, ;
   left(vl.property, 200) ;
FROM vinzari.dbc vl INNER JOIN vinzari.dbc v2 ;
   ON vl.parentid=v2.objectid ;
WHERE v2.OBJECTTYPE='View' AND vl.OBJECTTYPE='Field' ;
ORDER BY vl.objectid
```

8.3. Proceduri şi funcţii stocate în VFP şi Oracle

Dacă procedura sau funcţia reprezintă noţiuni vânturate de multe decenii de informaticienii cu sau fară acte în regulă, asocierea acestora cu persistenţa a fost consacrată de SGBD-urile deceniului 9 al tocmai încheiatului secol. Stocarea se referă la „înăuntrul” bazei, în dicţionarul de date sau catalogul sistemului, cum i se spune la case (de soft) mai mari.

Deşi puternic impregnate cu SQL, funcţiile şi procedurile stocate sunt, după cum bine le zice numele, cam... procedurale, depinzând într-o măsură decisivă de extensiile proprietare ale fiecărui SGBD. Dintre procedurile stocate, în acest paragraf ne ocupăm de funcţiile ce întorc valori implicite pentru atribute, proceduri/funcţii de validare, iar în paragraful următor, de un tip special de proceduri

stocate, şi anume declanşatoarele *(trigger-e*le).

Valori implicite

O problemă importantă a bazelor de date o reprezintă cheile-surogat, valori ale unor atribute care nu au legătură directă cu realitatea, dar ajută la identificarea de o manieră unică a liniilor unei tabele. Spre exemplu, în tabela CLIENŢI, atributul CodCl este o cheie- -surogat, fiind un număr unic atribuit fiecărui client, fără o logică anume. La fel putem spune despre CodPr din tabela PRODUSE şi Codlnc din ÎNCASĂRI. Din contra, CodPost din LOCALITĂŢI, Jud din JUDEŢE, CNP din PERSOANE, NrFact sunt informaţii cu alt grad de relevanţă şi standardizare.

Modul în care se pot genera valori implicite pentru atribute de tip chei-surogat diferă de la caz la caz. în Oracle, valorile secvenţiale unice pot fi generate printr-un tip special de obiecte ale bazei denumite „secvenţe", dar care nu pot fi referite în clauza DEFAULT, ci numai în declanşatoare.

DB2-ul permite, la crearea unei tabele, declararea de câmpuri autoincrementate, astfel:

```
CREATE TABLE clienţi (
   codci DECIMAL(6) NOT NULL
      GENERATED ALWAYS AS IDENTITY (START
      WITH 1000, INCREMENT BY 1), dencl
      VARCHAR(30)

);.
```

Orice linie inserată ulterior în tabela CLIENŢI va primi, pentru atributul CodCl, valoarea următoare, obţinută prin secvenţa declarată prin clauzele START WITH - INCREMENT BY.

Poate paradoxal, Visual FoxPro este mult mai generos în materie de funcţii stocate pentru calculul valorilor implicite ale unor atribute. Pentru atributul CodCl din tabela CLIENŢI se doreşte ca valorile implicite să fie „returnate" de funcţia def_codcl_clienti (). Comanda declarativă este:

```
ALTER TABLE CLIENŢI ALTER COLUMN CodCl ;
      SET DEFAULT def_codcl_clienti()
```

Procedurilor stocate ale bazei de date VÂNZĂRI (al cărei program de creare a fost discutat în capitolul 3) li se adaugă şi liniile din listingul 8.2.

Listing 8.2. Procedura stocată VFP6 DEF_C0DCL_CL1ENTI

```
**************************************
PROCEDURE def_codcl_clienti
'k'k-k-k'k-k'k'k'k-k'k-k-k-k-k'k-k-kic'k'k'k'fc-k'k-k'k-k'k-k'k-k-k'k'k'k
LOCAL vCodCl DIME
vCodCl(1,1)
vCodCl(1,1) = 0
SE.LECT MAX (CodCl) FROM clienţi INTO ARRAY vCodCl
IF vCodCl(1,1) = 0 RETURN 1001 ELSE
   RETURN vCodCl(1,1) + 1
ENDIF ENDPROC
```

Spre deosebire de alte SGBD-uri, în VFP funcţiile ce întorc valori implicite pot fi oricât de sofisticate. Lucrul acesta nu trebuie exagerat, deoarece poate afecta viteza de .lucru a aplicaţiei. Ne fixăm ca obiectiv crearea unei funcţii stocate care să determine Cel mai frecvent client, adică acel client care apare pe cele mai multe facturi, iar codul acestuia să constituie valoarea implicită pentru atributul CodCI din tabela FACTURI:

ALTER TABLE FACTURI ; .
ALTER COLUMN CodCI SET DEFAULT def_codcl_facturi()

In listingul 8.3 se află corpul funcţiei (în procedurile stocate ale bazei de date):

Listing 8.3. Corpul procedurii stocate VFP6

```
************************************
PROCEDURE def_codcl_facturi
**********************************
LOCAL vCodCl DIME vCodCl(1,2) vCodCl = 0
SELECT TOP 1 CodCI, COUNT(*) AS nr ;
FROM FACTURI ;
INTO ARRAY vCodCl ;
GROUP BY CodCI ;
ORDER BY nr DESC
IF vCodCl(1,1) = 0
   MESSAGEBOX('Nu se pot introduce facturi, '+CHR(13)+;
      'cita vreme nu exista nici un client !')
   RETURN .F.
ELSE
   RETURN vCodCl(1,1)
ENDIF
ENDPROC
```

Intrând şi mai mult în detalii, luăm în discuţie tabela LINIIFACT. Cheia primară a acesteia este combinaţia (NrFact, Linie). Trei atribute sunt susceptibile de a se procopsi cu valori implicite calculate prin funcţii stocate, NrFact, Linie şi CodPr. Logica acestora este diferită. Spre exemplu, este de presupus că factura curentă, pentru care se introduc linii, este cea mai recentă (are cel mai mare număr). Numărul liniei trebuie incrementat, iar produsul propus de funcţia stocată trebuie să fie cel mai frecvent facturat, însă nu trebuie să violeze unicitatea combinaţiei (NrFact, CodPr)!

```
ALTER TABLE LINIIFACT ALTER COLUMN NrFact ;
   SET DEFAULT def_nrfact_liniifact()
ALTER TABLE LINIIFACT ALTER COLUMN Linie ;
   SET DEFAULT def_linie__liniif act ()
ALTER TABLE LINIIFACT ALTER COLUMN CodPr ;
   SET DEFAULT def_codpr_liniifact()
```

Listingul 8.4. prezintă codul acestor trei funcţii stocate:

Listing 8.4. Alte trei proceduri stocate VFP6

```
************************************
PROCEDURE def_nrfact_liniifact
**********************************
LOCAL vNrFact_
DIME vNrFar" (1) vNrFact_ =
* se selecte:   cea mai recenta factura
SELECT MAX(Nruact) ;
FROM FACTURI ;
INTO ARRAY vNrFact_
IF vNrFact_(1,1) = 0
   MESSAGEBOX('Nu se pot introduce linii daca nu exista nici
   o; factura !')
   RETURN .F.
ELSE
   RETURN vNrFact_
ENDIF
```

```
ENDPROC
************************************
PROCEDURE def_linie_liniifact
************************************
LOCAL vLinie_, NrFact_
DIME vLinie_(1,1)
vLinie_ = 0
NrFact_ = LINIIFACT.NrFact
* se extrage ultima linie introdusa in factura
curenta SELECT MAX(Linie) ;
FROM LINIIFACT ;
INTO ARRAY vLinie_ ;
WHERE NrFact = NrFact
IF vLinie_(1,1) =0       && e prima linie din factura
   RETURN 1
ELSE                     && se incrementează
   RETURN vLinie_(1,1)  + 1
ENDIF
ENDPRO
C
************************************
PROCEDURE def_codpr_liniifact
************************************
LOCAL vCodPr_,
NrFact DIME
vCodPr_(1,2)
vCodPr_ = 0
NrFact_ = LINIIFACT.NrFact
** se selecteaza cel mai frecvent produs, exceptindu-le pe cele
care c' ** ar viola unicitatea combinaţiei (NrFact, CodPr)
SELECT TOP 1 CodPr, COUNT(*) AS nr ;
FROM LINIIFACT ;
 INTO ARRAY vCodPr_ ;
 WHERE CodPr NOT IN ;
    (SELECT CodPr ;
    FROM LINIIFACT ;
    WHERE NrFact = NrFact_ ) ;
 GROUP BY CodPr ;
 ORDER BY nr DESC
 IF vCodPr_(1, 1) = 0
    MESSAGEBOX('Nu se mai pot introduce produse pe aceasta ;
       factura !')
    RETURN .F.
 ELSE
    RETURN vCodPr_(1,1)
 ENDIF
 ENDPROC
```

Cât priveşte regulile de validare complexe, intenţiile generoase din SQL-92, de folosire a subconsultărilor în clauze CHECK, nu prea au fost preluate în SGBD-urile comerciale, însă logica acestora poate fi inclusă în totalitate în declanşatoare (*triggere*), pe care le vom discuta în următorul paragraf.

Poate că ar mai merita amintit că în VFP pot avea asociate reguli de validare nu numai atribute ale unor tabele, ci şi ale unor tabele derivate. Astfel, dacă dorim ca în tabela derivată vFacturi

atributele NrFact, Linie şi CodPr să primească aceleaşi valori implicite ca şi în tabela LINIIFACT, comenzile sunt:

```
=DBSETPROP('vFacturi.NrFact' , 'FIELD', 'DefaultValue',                    ;
   'def_nrfact_liniifact() ' )
=DBSETPROP('VFACTURI.Linie' , 'FIELD', 'DefaultValue', ;
   'def_linie_liniifact() ' )
=DBSETPROP('VFACTURI.CodPr', 'FIELD', 'DefaultValue', ;
   'def_codpr_liniifact()')
```

Funcţii stocate şi fraze SELECT în Visual FoxPro

în afara funcţiilor şi procedurilor stocate de tipul regulilor de validare, valorilor implicite sau declanşatoarelor, o facilitate importantă a VFP o constituie posibilitatea folosirii funcţiilor stocate în fraze SQL.

Să se afişeze, pentru fiecare client, valoarea vânzărilor, încasărilor şi restul de plată.

La soluţiile pur neprocedurale SQL adăugăm una care presupune crearea în baza de date a două funcţii, după cum urmează (listingul 8.5):

Listing 8.5. Două funcţii VFP stocate, utilizate în interogări

```
************ + *****-*****•**********
   PROCEDURE
   f_vinzari_cl
   PARAMETER codcl_
   LOCAL suma_
' DIME suma_(1,1)
   suma = 0
SELECT SUM(cantitate * pretunit * (1 + proctva)) ;
INTO ARRAY suma_ ;
FROM facturi f INNER JOIN liniifact lf ON f.nrfact=lf.nrfact ;
   INNER JOIN produse p ON lf.codpr=p.codpr ;
WHERE codcl_ = codcl
RETURN suma__
ENDPROC
**********
**********
PROCEDURE f__incasari_cl PARAMETER codcl_
LOCAL suma_
DIME suma_(1,1) suma_ = 0
SELECT SUM(transa) ;
INTO ARRAY suma_ ;
FROM facturi f INNER JOIN  incasfact i ON f.nrfact=i.nrfact ;
WHERE codcl_ = codcl
RETURN suma_
ENDPROC
**********
```

Cele două funcţii pot fi folosite în orice frază SELECT după cum urmează:

```
SELECT dencl, ;
   f_vinzari_cl    (codcl) AS vinzari, ;
```

```
   f_incasari__cl (codcl) AS incasari, ; f_vinzari_cl
                  (codcl) - f__incasari_cl  (codcl)  ;
     AS rest_de_plata ;
FROM CLIENŢI
```

Care este cel mai mare datornic dintre clienţi?

Este una dintre problemele cele mai dificile formulate în capitolul 5, la care în VFP, singurele soluţii se bazau pe salvarea rezultatelor intermediare în cursoare. Iată o altă interogare, mult mai simplă, care se foloseşte de aceleaşi două proceduri stocate:

```
SELECT dencl, ;
   f_vinzari_cl    (codcl) AS vinzari, ;
   f_incasari_cl (codcl) AS incasari, ; f_vinzari_cl
                  (codcl) - f__incasari_cl  (codcl)  ;
       AS rest_de_plata ;
FROM CLIENŢI ;
WHERE f_vinzari_cl (codcl) - f_incasari_cl (codcl) = ; (SELECT
MAX(f_vinzari_cl (codcl) - ;
       f_incasari_cl (codcl)) ;
FROM CLIENŢI)
```

Nu este neapărat ca funcţiile de tipul `f_vinzari_cl` şi `f__incasari_cl` să fie stocate. Frazele SELECT pot fi incluse în programe ce includ şi descrierea funcţiilor-argument. în acest mod se evită aglomerarea containerului (dicţionarului) bazei.

Care este contribuţia procentuală a fiecărui produs la totalul vânzărilor?

După cura observaţi, am trecut deja la OLAP. Răspunsul la această întrebare presupune folosirea interogărilor scalare în DB2 şi a funcţiilor OLAP în Oracle 8i2. Varianta VFP pe care v-o propunem se bazează pe două funcţii stocate, prezentate în listingul 8.6.

Listing 8.6. Funcţii stocate pentru calculul vânzărilor pe produse şi vânzărilor totale

```
PROCEDURE
vinz_prod
PARAMETER codpr_
LOCAL suma_
DXME suma_(l,l)
suma_ = 0
SELECT SUM(cantitate * pretunit * (1 + proctva))          ;
INTO ARRAY suma_ ;
FROM liniifact lf INNER JOIN produse p ON lf.codpr=p.codpr ;
WHERE lf.codpr = codpr_
RETURN suma_
ENDPROC
**************************************


PROCEDURE total_vinzari
LOCAL suma_
DIME suma_(1,1)
suma_ = 0
SELECT SUM(cantitate * pretunit * (1 + proctva)) ;
INTO ARRAY suma_ ;
FROM liniifact lf INNER JOIN produse p ON lf.codpr=p.codpr
RETURN suma_
```

```
ENDPROC
**************************************
```

Iată şi interogarea care le foloseşte şi furnizează răspunsul la problema formulată:

```
SELECT denpr AS Produs, vinz_prod (codpr) AS Vinzari, ;
   total_vinzari() AS total, ;
   ROUND(vinz_prod (codpr) / total_vinzari() * 100,0)      ;
      AS Procent ;
FROM PRODUSE
```

Care este evoluţia zilnică a vânzărilor, prin raportare la ziua calendaristică anterioară?

Este nevoie de două funcţii, al căror corp este prezentat în listingul 8.7.

Listing 8.7. Funcţii pentru calculul vânzărilor unei zile şi zilei precedente

```
'k'k'k'k'k-k'k-k-k'k'k-k'k'k'k'k'k'k'k'k'k-k'k'k'k'k'k'k'k'k'k PROCEDURE
f_vinzari_zi PARAMETER data_
LOCAL suma_
DIME suma_(1,1)
suma_ = 0
SELECT SUM(cantitate * pretunit *     (1 + proctva)) ;
   INTO ARRAY suma_ ;
   FROM facturi f INNER JOIN liniifact lf ;
        ON f.nrfact=lf.nrfact ;
      INNER JOIN produse p ON lf.codpr=p.codpr ;
   WHERE datafact = data_
   RETURN suma_
   ENDPROC

   'k-k'k'k-k-k'k-k-k'k-k-k-k'k'k-k'k'k-k-k'k'k-k-k-k'k-k-k-k'k-k + 'k'k'k'k •k'k-k-k'k-k-k-k'k'k'k'k'k-

   k-k'k-k-k'k-k'k-k'k'k'k-k-k-k-k-k'k'k'k'k'k PROCEDURE f_vinzari_zi_prec

   PARAMETER data_
   RETURN f_vinzari_zi.(data_ - 1)
   ENDPROC

   ▪k-k'k-k'k-k'k'k-k-k'k'k'k-k-k-k-k-k'k'k'k'k'k'k'k-k-k'k'k'k'k-k-k-k'k'k-k în final, iată
```

şi interogarea:

```
SfiLECT DISTINCT datafact AS Zi, ;                          _
   f_vinzari_zi (datafact) AS Vinzari_Zi_Curenta,          ;
   f_vinzari_zi_prec   (datafact)   AS   Vinzari_Zi_Precedenta,   ;
   f_vinzari_zi (datafact) - f_vinzari_zi_prec (datafact) ;
      AS Diferenţa ;
FROM FACTURI
```

Sigur, acest stil de lucru poate oripila pe mulţi ideologi şi convertiţi la SQL, dar până la apariţia interogărilor scalare şi funcţiilor OLAP în Visual FoxPro nu prea avem de ales...

Funcţii stocate şi fraze SELECT în Oracle

Modul de lucru prezentat mai sus funcţionează şi în Oracle. Este adevărat, în Oracle se poate discuta despre nivelul de puritate al funcţiilor, obligatoriu pentru cele incluse în pachete. Ca model, prezentăm în listingul 8.8 scriptul de creare a celor două funcţii şi echivalentele funcţiilor VFP cu

aceleaşi nume din listingul 8.5.

Listing 8.8. Crearea în Oracle a funcţiilor stocate, utilizate în interogări

```
CREATE OR REPLACE FUNCTION f_vinzari_cl (
   codcl_ IN clienţi.codcl%TYPE ) RETURN NUMERIC
IS
   suma_ NUMERIC (16) ;
BEGIN '
   SELECT SUM(cantitate * pretunit * (1 + proctva))
   INTO suma__
   'FROM FACTURI F, LINIIFACT LF, PRODUSE P WHERE
   F.NrFact = LF.NrFact AND LF.CodPr = P.CodPr AND
   codcl_ = codcl ;
   RETURN suma_ ;
END ;
```

```
CREATE OR REPLACE FUNCTION f_incasari_cl (
   codcl_ IN clienţi.codcl%TYPE ) RETURN NUMERIC
IS
   suma_ NUMERIC (16) := 0 ;
BEGIN '
   SELECT SUM(transa)
   INTO suma_ '
   FROM FACTURI F, INCASFACT I
   WHERE F.NrFact = I.NrFact AND CodCI = CodCl_ ;
   RETURN suma_ ;
END ;
/
```

Cât priveşte interogarea care le foloseşte, nu sunt deosebiri faţă de VFP:

```
SELECT dencl,
   f_vinzari_cl     (codcl)     AS     vinzari,
   f_incasari_cl    (codcl)    AS    incasari,
   f_vinzari__cl (codcl) - f_incasari_cl (codcl)
      AS rest_de_plata
FROM CLIENŢI
```

Ar mai fi de adăugat că pentru a afla numele şi alte informaţii despre funcţiile şi procedurile stocate din schema curentă, se poate interoga catalogul sistem:

```
SELECT *
FROM USERjDBJECTS
WHERE OBJECT_TYPE = 'FUNCTION'
```

respectiv:

```
SELECT *
FROM USERJDBJECTS
WHERE OBJECT_TYPE = 'PROCEDURE'
```

în schimb, corpul funcţiilor/procedurilor este conţinut în altă tabelă a dicţionarului - USER_SOURCE. Astfel, pentru a vizualiza conţinutul funcţiei AF_DENCL, este necesara consultarea:

```
SELECT line, text FROM USER_SOURCE
WHERE TYPE = 'FUNCTION' AND NAME='AF_DENCL'
```

în DB2, informaţii despre funcţiile stocate într-o schemă (FOTACHEM) pot fi vizualizate prin interogarea:

```
SELECT *
FROM SYSCAT.FUNCTIONS

WHERE FUNCSCHEMA='FOTACHEM'
```

■

Analog stau lucrurile cu procedurile stocate:

```
SELECT *
FROM SYSCAT.PROCEDURES
WHERE
PROCSCHEMA='FOTACHEM'
```

8.4, Declanşatoare în VFP şi Oracle

Declanşatoarele (*triggere*, în original) reprezintă un tip deosebit de proceduri stocate ale unei baze de date. Particularitatea lor esenţială ţine de faptul că, o dată create şi stocate în schema bazei, acestea

sunt executate automat ia operaţiuni de inserare, modificare sau ştergere a liniilor din tabela pentru care au fost definite. La această familie a declanşatoarelor, unele SGBD-uri au mai adăugat şi alte tipuri, asociate, spre exemplu, actualizărilor tabelelor derivate, conectării unei aplicaţii/utilizator, deschiderii şi închiderii instanţei baze de date etc.

Pentru lucrarea de faţă, problema principală a declanşatoarelor ţine de faptul că acestea sunt scrise în extensiile procedurale ale SQL care prezintă diferenţe sensibile de la produs la produs: PL/SQL (Oracle), DB2, Transact-SQL (SQL Server) etc. Este drept că în SQL-3 există o parte special dedicată procedurilor stocate, dar preluarea pe scară largă a specificaţiilor standardului în produsele marilor actori ai pieţei SGBD-urilor va lua ceva timp.

Cele mai importante avantaje ale declanşatoarelor sunt:

- permit instituirea unor reguli de validare cu mult mai complexe decât ceea ce permite clauza CHECK;
- pot ameliora sensibil mecanismul de securitate al bazei;
- permit instituirea (acolo unde nu este posibil prin clauza FOREIGN KEY) restricţiilor referenţiale şi modului de tratare a modificărilor ce pot cauza probleme de integritate referenţială;
- constituie suportul calculării automate a valorilor unor atribute.

într-o abordare de la simplu la complex, vom urmări câteva aspecte ale declanşatoarelor în Visual FoxPro 6 şi Oracle 8i.

Declanşatoare în VFP6

Problematica declanşatoarelor în VFP6 am prezentat-o *in extenso* într-un articol publicat în *PC Report* împreună cu Cătălin Strîmbei[62]. Fără a intra prea mult în detalii, în VFP există trei tipuri de declanşatoare: pentru inserare, modificare şi ştergerea de linii. Momentul declanşării lor depinde esenţial de tipul de Buf fering ales, o chestiune specifică VFP. în cele ce urmează, presupunem că opţiunea de *huffering* este dezactivată.

Declanşatoare pentru restricţii referenţiale

Mecanismul de asigurare a restricţiilor referenţiale se poate implementa grafic cu ajutorai modulului Referenţial Integrity Builder. Paradoxal, deşi există clauza FOREIGN KEY, aceasta instituie în VFP numai legături permanente între tabele. De aceea, la crearea prin program a bazei de date sunt necesare declanşatoare „manuale", care, totuşi, au avantajul că permit afişarea unor mesaje de eroare în clar sau afişarea unor restricţii şi explicaţii suplimentare.

Să începem cu tabela JUDEŢE. Ştergerea unui judeţ poate aduce necazuri dacă există localităţi în acel judeţ (mă refer la baza de date...). Aşa încât este necesar un declanşator de ştergere care să verifice dacă eliminarea înregistrării poate avea loc sau nu. Comanda de creare a declanşatorului este:

CREATE TRIGGER ON judeţe FOR DELETE AS trg_del_judete()

Iată, în listingul 8.9, corpul declanşatorului.

Listing 8.9. Varianta 1 a declanşatorului TRG_DEL_JUDETE

```
************************************
PROCEDURE trg_del_judete
************************************
LOCAL jud_, v
dime v(1,1) v
= '    •
```

62. [Fotache&Strîmbei99-1].

```
jud_ = judeţe.jud
SELECT MIN(loc) FROM localitati INTO ARRAY v WHERE jud = jud_
IF v(1,1) # '  '
   MESSAGEBOX(1 Exista cel puţin o linie copil in ;
      tabela LOCALITATI.'+;
      CHR(13)+'Exemplu: loc. '+ALLTRIM(v(1,1)))
   RETURN .F.
ENDIF
ENDPROC
************************************
```

SQL beneficiază de câteva optimizări ale motorului bazei date. Cu toate acestea, SELECT-ul din declanşator parcurge întreaga tabelă LOCALITĂŢI, aşa încât pare mai rapidă varianta următoare - ce-i drept, mai puţin SQL-istă.

Listing 8.10. Varianta 2 a declanşatorului TRG_DEL_JUDETE

```
**************************************
PROCEDURE trg_del_judete
************************************
LOCAL jud_ jud_ =
judeţe.jud
IF INDEXSEEK(jud_, .F., 'localitati', 'jud')
        MESSAGEBOX('Exista cel puţin o linie copil in ;
                tabela LOCALITATI !')
        RETURN .F.
ENDIF
ENDPROC
**************************************
```

Funcţia INDEXSEEK caută în tabela LOCALITĂŢI, fară a deplasa pointerul pentru înregistrări, prima linie pentru care cheia indexului JUD (cheie care este atributul cu acelaşi nume) are aceeaşi valoare ca a variabilei j ud_. Când căutarea este încununată de succes, valoarea retumată este TRUE (. T. în notaţia VFP), altminteri, FALSE.

Modificarea indicativului unui judeţ trebuie să se propage şi în tabela LOCALITĂŢI, ocazie bună pentru încă un declanşator:

```
CREATE TRIGGER ON judeţe FOR UPDATE AS trg_upd_judete()
```

Corpul declanşatorului este prezentat în listingul 8.11. Valoarea dinaintea modificării se obţine în VFP prin funcţia `OLDVAL ()`.

Listing 8.11. Declanşator pentru modificarea unei linii din tabela JUDEŢE

```
-k-k-k-k-k-k-k-k-k-k-k-k-k-k-kie-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k-k
PROCEDURE trg_upd_judete
**************************************

LOCAL jud_OLD, jud_NEW
j ud_NEW = judeţe.jud
jud_OLD = OLDVAL('jud', 'judeţe')
IF jud_NEW # jud_OLD
   UPDATE localitati SET jud = jud_NEW WHERE jud =
jud_OLD ENDIF ENDPROC
```

La inserarea unei noi linii într-o tabelă-copil sau la schimbarea valorii unei chei străine, trebuie verificat dacă noile valori se regăsesc în tabela-părinte. Spre exemplu, dacă în tabela LOCALITĂŢI se modifică valoarea atributului `Jud`, această nouă valoare trebuie căutată în JUDEŢE. Se poate crea, în acest scop, un declanşator:

```
CREATE TRIGGER ON localitati FOR UPDATE ;
   AS trg_upd_localitati()
```

Trebuie, însă, să avem în vedere că modificarea atributului `Jud` din LOCALITĂŢI poate fi şi consecinţa declanşatorului `trg_upd_judete`. Aşa încât, pentru a evita circularitatea, se verifică dacă modificarea este una locală sau provine din declanşatorul cu pricina - vezi listingul 8.12.

Listing 8.12. Declanşator TRG_UPD_LOCALITATI

```
**************************************
PROCEDURE trg_upd_localitati
**********************************
LOCAL jud_OLD, jud_NEW, codpost_OLD, codpost NEW, v ;
DIME v(1,1)                                        ~
*** mai intii se verifica restrictia referentiala cu tabela părinte
jud_NEW = localitati.jud
jud_OLD = OLDVAL('jud', 'localitati')

IF jud_NEW # jud__OLD
   IF _TRIGGERLEVEL <= 1   && modificarea este locala
      SELECT jud FROM judeţe INTO ARRAY v WHERE jud =
      jud_NEW IF _TALLY = 0
         MESSAGEBOX(1 Noua valoarea a atributului Jud ;
            - '+j ud_NEW+CHR(13)+;
            ' nu se regaseste in tabela părinte JUDEŢE !')
         RETURN .F.
      ENDIF
   ENDIF
ENDIF  ***  se  propaga  eventuala  modificare  a  CodPost  in  tabela
CLIENŢI
codpost_NEW = localitati.codpost
codpost_OLD = OLDVAL('codpost', 'localitati')
```

```
IF codpost_NEW # codpost_OLD
   UPDATE clienţi SET codpost = codpost_NEW ;
     WHERE codpost = codpost__OLD
ENDIF
ENDPROC
```

Variabila sistem _TRIGGERLEVEL indică tocmai dacă avem de-a face cu o modificare în cascadă sau una locală a atributului `Jud`. Ca un alt element de noutate, funcţia sistem TALLY întoarce numărul de linii extrase prin clauza WHERE a ultimei comenzi SQL (SELECT). Pentru a accelera căutarea în tabela-părinte, se poate folosi în locul SELECT-ului funcţia INDEXSEEK.

Atribute actualizate prin declanşatoare

Să luăm în discuţie două speţe. în paragraful 6.5 am adăugat câmpul `ValTotală` tabelei FACTURI, despre care spuneam că este un atribut calculat şi actualizabil prin declanşatoare. Valoarea totală a unei facturi este dată de atributele `Cantitate` şi `PretUnit` din tabela LINIIFACT şi atributul `ProcTVA` din PRODUSE.

Trecem peste declanşatoarele tabelei PRODUSE şi îl prezentăm numai pe cel corespunzător actualizării fiecărei linii a tabelei LINIIFACT.

```
CREATE TRIGGER ON liniifact FOR UPDATE AS trg_upd_liniifact()
```

Acesta are mai multe sarcini:

* verifică dacă s-a produs o modificare a atributului `NrFact` şi dacă eventuala valoare nouă se regăseşte în tabela FACTURI;
- dacă s-a modificat `CodPr`, se testează existenţa noii valori în PRODUSE;
« dacă s-a modificat `Cantitate` sau `PretUnit`, se actualizează valoarea atributului `ValTotala` în tabela FACTURI.

în listingul 8.13 este prezentat corpul declanşatorului:

Listing 8.13. Declanşatorul VFP pentru modificarea unei linii din tabela LINIIFACT

```
**********************************
PROCEDURE trg_upd_liniifact
**********************************
LOCAL nrfact_OLD, nrfactJMEW, codpr_OLD, codpr_NEW, v, ;
   cantitate_OLD, cantitate_NEW, pretunit_OLD, pretunit_NEW,
   ; proctva_OLD, proctva_NEW
cantitate_OLD = OLDVAL('cantitate',
'liniifact') cantitate_NEW =
liniifact.cantitate pretunit_OLD =
OLDVAL('pretunit', 'liniifact') pretunit_ NEW =
liniifact.pretunit
DIME v(l,l)
*** se verifica restrictia referentiala cu tabela părinte FACTURI
nrfact_NEW = liniifact.nrfact
nrfact_OLD = OLDVAL(1nrfact', 'liniifact')
IF nrfact_NEW # nrfact_OLD
   IF _TRIGGERLEVEL <= 1   && modificarea este locala
      IF !INDEXSEEK(nrfact_NEW, .F., 'facturi', 'nrfact')
         MESSAGEBOX('Noua valoare a atributului NrFact
            +LTRIM(STR(nrfact_NEW,8)) + ;
           CHR(13)+'nu exista in tabela părinte FACTURI !')
```

```
        RETURN .F.
      ENDIF                                    ■
   ENDIF
ENDIF
*** se verifica restrictia referentiala cu tabela părinte PRODUSE
codpr_NEW = liniifact.codpr
codpr_OLD = OLDVAL('codpr', 'liniifact')
IF codpr_NEW # codpr_OLD
   IF __TRIGGERLEVEL <= 1  && modificarea este locala
      IF !INDEXSEEK(codpr_NEW, .F., 'produse', 'codpr')
         MESSAGEBOX('Noua valoare a atributului CodPr
            +LTRIM(STR(codpr_NEW,6)) + ;
           CHR(13)+'nu exista in tabela părinte PRODUSE !')
         RETURN .F.
      ENDIF
   ENDIF
ENDIF
 *** Daca s-a schimbat numărul facturii, linia trebuie scoasa de la
* fosta factura si adaugata la noua factura IF nrfact_NEW #
nrfact_OLD AND _TRIGGERLEVEL <~ 1 v = .00
   SELECT ProcTVA FROM produse INTO ARRAY v ;
      WHERE CodPr = CodPr_OLD
   proctva_OLD = v(l,l) v =
   .00
   SELECT ProcTVA FROM produse INTO ARRAY v ;
      WHERE CodPr = CodPr_NEW
   proctva_NEW = v(l,l)
   * se scade valoarea de la factura "veche"
   UPDATE facturi SET valtotala = valtotala - ;
        cantitate_OLD * pretunit_OLD *  (1     +      proctva_OLD) ;
   WHERE nrfactura = nrfactura_OLD
   * se aduna valoarea de la factura "noua"
   UPDATE facturi SET valtotala = valtotala + ;
        cantitate_NEW * pretunit_NEW *  (1  +  proctva_NEW) ;
   WHERE nrfactura = nrfactura_NEW
ELSE ** nu s-a modificat NrFact
   ** Daca s-a modificat codul produsului, deci trebuie văzut
   ** daca procentul TVA este acelaşi IF codpr_NEW #
   codpr_OLD v = . 00
      SELECT ProcTVA FROM produse INTO ARRAY v ;
        WHERE CodPr = CodPr_OLD proctva_OLD =
      v(l,l) v = .00
      SELECT ProcTVA FROM produse INTO ARRAY v ;
        WHERE CodPr = CodPr_NEW proctva_NEW =
      v(l,l)
      IF proctva_NEW # proctva_OLD
        UPDATE facturi SET valtotala = valtotala +
           ; cantitate_NEW * pretunit_NEW * ;
           (1 + proctva_NEW) - ;
           cantitate_OLD * pretunit_OLD *
           ;
           (1 + proctva_OLD) ;
        WHERE nrfact = nrfact_OLD ENDIF
   ELSE && nu s-a schimbat CodPr
```

```
      IF cantitate_NEW # cantitate_OLD OR ;
      pretunit_NEW # pretunit_OLD v = .00
         SELECT ProcTVA FROM produse INTO ARRAY v ;
            WHERE CodPr = CodPr_OLD
         proctva_OLD = v(1,1)
         UPDATE facturi SET valtotala = valtotala +
            ; cantitate_NEW * pretunit_NEW * ;
            (1 + proctva_OLD) - ;
            cantitate_OLD * pretunit_OLD *
            ;
            (1 + proctva_OLD) ;
         WHERE nrfact = nrfact_OLD ENDIF
   ENDIF
ENDIF
ENDPROC
************************************
```

Declanşatoare pentru controlul actualizărilor

Să imaginăm două situaţii în care s-ar cuveni să restricţionăm operaţiunile de actualizare a tabelei FACTURI. Prima ţine de interzicerea modificării atributului ValTotala altfel decât prin declanşatoarele tabelei LINIIFACT. în lipsa unei asemenea opţiuni, orice utilizator poate să modifice direct (prin BROWSE, REPLACE sau UPDATE) valoarea totală a unei facturi, creându-se astfel o diferenţă faţă de informaţiile din tabela LINIIFACT. Ideea este simplă: atunci când se modifică atributul ValTotala, se verifică dacă nivelul declanşatorului este mai mare decât 1, ceea ce ar constitui un indiciu că modificarea provine din unul dintre *trigger-e*le tabelei LINIIFACT. în această situaţie, modificarea este autorizată, în caz contrar, interzisă.

A doua problemă se referă la o restricţie încadrabilă în categoria *reguli ale afacerii (.Business Rules*): se interzic noi vânzări pentru clienţii care au un rest de plată mai mare de 100 milioane de lei. Problema trebuie rezolvată prin două declanşatoare, unul pentru inserare şi altul de actualizare. în cele ce urmează îl prezentăm numai pe cel de actualizare.

Corpul declanşatorului este prezentat în listingul 8.14. Listing 8.14. Declanşatorul pentru modificarea unei linii din tabela FACTURI

```
************************************
PROCEDURE trg_upd_facturi ************************************
LOCAL codcl_OLD, codcl_NEW, valtotala_OLD, valtotala_NEW, ;
  nrfact_OLD, nrfact_NEW, vinzari_cl, incasari cl, v DIME
v(1,1) v = 0
*** se verifica restrictia referentiala cu tabela părinte CLIENŢI
codcl_NEW = facturi.codcl
codcljDLD = OLDVAL('codcl', 'facturi')
IF codcl_NEW # codcl_OLD
  IF _TRIGGERLEVEL <= 1   && modificarea este locala
     IF !INDEXSEEK(codcl_NEW, .F., 'clienţi', 'codcl')
        MESSAGEBOX(1 Noua valoare a atributului CodCI
           +LTRIM(STR(codcl_NEW,6)) + ;
          CHR(13)+'nu exista in tabela părinte CLIENŢI !')
        RETURN .F.
     ENDIF
```

```
  ENDIF
ENDIF
** daca se modifica CodCI, trebuie văzut daca noul client are **
datorii mai mari de 100000000 (lei)
IF codcl_NEW # codcl OLD v=0
  SELECT SUM(Cantitate * PretUnit * (1 + ProcTVA)) ;
  INTO ARRAY v ;
  FROM facturi f INNER JOIN liniifact lf ;
       ON f.nrfact = lf.nrfact ;
     INNER JOIN produse p ON lf.codpr = p.codpr ;
  WHERE codcl = codcl_NEW
  vinzari_cl = v(1,1)
  v=0
  SELECT SUM(Transa) ;
  INTO ARRAY v ;
  FROM facturi f INNER JOIN incasfact i ON f.nrfact = i.nrfact ;
  WHERE codcl = COdd_NEW incasari_cl = v(1,1)
  IF vinzari_cl - incasari_cl > 100000000
    MESSAGEBOX('Noul client e prea datornic si ;
      nu-i mai vindem nimic !!!')
    RETURN .F.
  ENDIF
ENDIF
*** se interzice modificarea interactiva a atributului ValTotala
valtotala__NEW = facturi . valtotala valtotala_OLD =
OLDVAL('valtotala', 'facturi')
IF valtotala_NEW # valtotala_OLD
  IF _TRIGGERLEVEL <= 1
    MESSAGEBOX('Nu puteti modifica interactiv valoarea totala ;
      a facturii ! ')
    RETURN .F.
  ENDIF
ENDIF
*** se propaga eventuala modificare a NrFact in tabelele copil -
*** LIINIIFACT si INCASFACT
nrfact_NEW = facturi.nrfact
nrfact_OLD = OLDVAL('nrfact', 'facturi')
IF nrfact_NEW # nrfact_OLD
  UPDATE liniifact SET NrFact = nrfact_NEW ;
    WHERE NrFact = nrfact_OLD
  UPDATE incasfact SET NrFact = nrfact_NEW ;
    WHERE NrFact = nrfact_OLD
ENDIF
ENDPROC
```

Toate procedurile stocate (codul acestora) se găsesc în atributul Code de pe linia asociată obiectului StoredProceduresSource.

Declanşatoare în Oracle 8i

Aria de utilizare a *trigger-e*lor în Oracle este şi mai extinsă: restricţii complexe, asigurarea integrităţii BD, îmbunătăţirea securităţii, respectarea integrităţii referenţiale între nodurile unei BD distribuite, jumalizarea operaţiunilor (tranzacţiilor), sincronizarea replicilor unei BD etc.

Totuşi, documentaţia Oracle 8 recomandă ca elanul *trigger-e,*sc să nu depăşească anumite limite. Prin declanşatoare trebuie gestionate numai acele reguli imposibil de asigurat prin restricţiile la nivel de atribut (câmp), înregistrare, restricţii de tip PRIMARY KEY, NOT NULL, SET DEFAULT etc. în plus, *Application Developer's Guide* specifică un număr maxim de linii ce ar trebui să alcătuiască un *trigger* - 60, la nevoie fiind indicată crearea unor proceduri stocate apelabile din *trigger.*

Logic, pentru cel considerat de mulţi practicieni drept cel mai bun SGBD la ora actuală (dar şi unul dintre cele mai scumpe), Oracle oferă o tipologie a declanşatoarelor mult mai bogată decât VFP. în VFP descriam trei tipuri, aferente celor trei operaţiuni ce desemnează actualizarea unei tabele: inserare, modificare şi ştergere. în funcţie şi de tipul de *buffering* ales, declanşatoarele intră în acţiune DUPĂ inserare, modificare sau ştergere, iar printr-un RETURN . F., plasat în corpul *trigger*-ului, operaţiunea este anulată, revenindu-se la starea dinainte.

în Oracle 8 există 15 tipuri diferite de declanşatoare - vezi tabelul 8.1.

Tabelul 8.1. Tipologia „clasică" a declanşatoarelor Oracle

Eveniment declanşator	Declanşare pt. fiecare	Descriere
BEFORE INSERT	Comandă	Codul (programul) acestuia se lansează înaintea executării unei comenzi INSERT în tabela-ţintă
BEFORE INSERT	Linie	Se execută înaintea inserării fiecărei linii în tabelă
AFTERINSERT	Linie	Se execută după inserarea fiecărei linii în tabelă
AFTERINSERT	Comandă	Se lansează după execuţia unei comenzi de inserare de linii în tabelă
INSTEAD OF INSERT	Linie	în loc să se insereze o linie în tabelă, se execută codul din acest declanşator
BEFORE UPDATE	Comandă	Codul acestuia se execută înaintea executării unei comenzi UPDATE pentru tabela-ţintă
BEFORE UPDATE	Linie	Se execută înaintea modificării fiecărei linii din tabelă
AFTER UPDATE	Linie	Se execută după modificarea fiecărei linii
AFTER UPDATE	Comandă	Se lansează după execuţia comenzii UPDATE (după modificarea tuturor liniilor afectate de comandă)
INSTEAD OF UPDATE	Linie	în loc să se modifice o linie din tabela-ţintă, se execută codul din acest declanşator
BEFORE DELETE	Comandă	Se execută înaintea comenzii DELETE
BEFORE DELETE	Linie	Se execută înainte de ştergerea fiecărei linii
AFTER DELETE	Linie	Se execută după ştergerea fiecărei linii
AFTER DELETE	Comandă	Se lansează după execuţia comenzii DELETE (după ştergerea tuturc'- 'iniilor afectate de comandă)
INSTEAD OF DELETE	Linie	Se execută în locul ştergerii unei linii din tabela- 1 -tintă

O dată cu versiunea 8i, în Oracle au mai fost introduse o serie de declanşatoare lansabile în

execuţie la deschiderea instanţei bazei, închiderea instanţei bazei, conectarea şi deconectarea utilizatorilor/aplicaţiilor etc.

Declanşatoare la nivel de linie şi la nivel de comandă (statement)

La definirea *trigger*-ului se poate stabili de câte ori este executat acesta: de fiecare dată când o linie este inserată/modificată/ştearsă, sau o singură dată pentru o comandă de actualizare, indiferent de câte linii sunt afectate.

Să luăm, spre exemplu, comanda:

```
UPDATE clienţi SET codcl = codcl + 100 WHERE codcl > 3
```

Presupunând că 135 de clienţi au codul mai mare decât 3, un *trigger* de modificare la nivel de linie se execută de 135 de ori, în timp ce unul la nivel de comandă o singură dată.

Dacă nici un client nu are codul mai mare de 3, un *trigger* de modificare la nivel de linie nu s-ar putea executa deloc, în timp ce declanşatorul declarat a fi la nivel de comandă ar fi lansat (evident, o singură dată).

Declanşatoare de tip înainte (BEFORE) şi după (AFTER) actualizare

O altă diferenţiere (în bine) faţă de declanşatoarele VFP este posibilitatea de a defini momentul acţiunii *trigger*-ului. Un *trigger* de tip BEFORE intră în acţiune înainte de modificarea propriu-zisă şi este ideal pentru verificarea unui set de condiţii care ar putea bloca de la bun început tranzacţia respectivă (autentificări, soldul 0 pentru un cont sau pentru un material într-o magazie etc.). Declanşatorul de tip AFTER se execută după momentul producerii actualizării, servind la verificarea corectitudinii operaţiunii (nu a fost scoasă din magazie o cantitate de material mai mare decât stocul dinaintea operaţiunii), inserarea unei linii într-o tabelă de tip jurnal (ce ţine evidenţa strictă a modificărilor operate în tabele cu informaţii sensibile - spre exemplu, calculul automat al costului mediu ponderat al unui material după fiecare intrare în magazie sau calculul soldului curent al unui cont bancar după fiecare depunere sau retragere etc.).

Spre deosebire de cele de tip BEFORE, *trigger-e*le de tip AFTER blochează liniile procesate.

Declanşatoarele la nivel de linie/comadă pot fi combinate cu cele de tip BEFORE/ AFTER. Astfel, pentru orice tabelă şi operaţie de inserare/modificare/ştergere pot fi definite patru tipuri de *triggere* care se execută în ordinea:

- înainte - la nivel de comandă (BEFORE - statement)
- înainte - la nivel de linie (BEFORE - row)
- după-la nivel de linie (AFTER-row)
- după - la nivel de comandă (AFTER - statement).

Triggere *de tip în-loc-de (INSTEAD OF)*

Acest gen de declanşator constituie un excelent mijloc de propagare a modificărilor dintr-o tabelă derivată în tabelele de bază din care provine. Amânăm această discuţie pentru finalul capitolului.

Un alt element foarte important este că şi în *trigger-e* le la nivel de linie de tipBEFORE, şi în cele de tip AFTER pot fi invocate şi prelucrate atât valorile atributelor dinaintea operaţiei, cât şi cele de după operaţie. Prefixul este, după caz, : OLD sau : NEW (vă amintiţi că în VFP aveam nevoie de funcţia OLDVAL ()).

Declanşatoare pentru generarea cheilor-surogat

După cum am mai discutat, în Oracle nu pot fi definite, nici pentru valorile implicite, nici pentru regulile de validare, funcţii-utilizator. Lucrurile pot fi însă aranjate utilizând declanşatoare. Spre exemplu, pentru ca un client nou să primească codul următor, se poate declara un declanşator la nivel de linie prin care, înaintea inserării noii linii, valoarea `CodCI` să fie obţinută de o manieră similară VFP - vezi listingul 8.15.

Listing 8.15. Declanşatorul pentru inserarea unei înregistrări în tabela CLIENŢI - varianta 1

```
CREATE OR REPLACE TRIGGER trg_clienti ins befo row BEFORE INSERT ON
clienţi FOR EACH ROW DECLARE
   cc_ clienţi.codclITYPE ;
BEGIN
   SELECT MAX(codcl)
   INTO cc
   FROM clienţi ;
   :NEW.codcl := NVL(cc_,1000) + 1 ;
END ;
```

Chiar dacă comanda de inserare are forma:

```
INSERT INTO clienţi VALUES (999, 'Client 8', NULL,
    NULL, '6600', NULL) ;
```

valoarea atributului `CodCl` în noua linie nu va fi 999, ci calculată prin declanşator.

O altă variantă a acestui declanşator utilizează secvenţele. Secvenţa permite unei aplicaţii să preia numere consecutive unice pe un anumit interval. Gestiunea secvenţei se face la nivelul central; fiecare apel care „cere" o valoare din secvenţă poate fi absolut sigur că nici un alt apel nu va primi o valoare identică, deci nu vor apărea conflicte. O secvenţă se creează prin comanda CREATE SEQUENCE. Pentru ca noii clienţi să primească coduri unice, în ordinea preluării în tabelă, se poate folosi secvenţa seq_clienti_codcl creată după cum urmează:

```
CREATE SEQUENCE seq_clienti codcl INCREMENT BY 1 MINVALUE 1010 MAXVALUE 5555
    NOCYCLE
    ORDER                                                              .
```

O dată creată, referinţa la o secvenţă se face prin NextVal, caz în care se obţine valoarea următoare a secvenţei (incrementarea se face automat), în timp ce valoarea curentă a secvenţei se obţine prin CurrVal.

Astfel încât, pentru ca un client nou să aibă codul imediat superior ultimului client introdus în tabelă, se foloseşte declanşatorul trg_clienti ins befo row, care se execută automat înaintea adăugării unei noi înregistrări în tabela CLIENŢI.

Listing 8.16. Declanşator pentru inserarea unei linii în tabela CLIENŢI - varianta 2

```
CREATE OR REPLACE TRIGGER trg_clienti ins befo row BEFORE INSERT ON
clienţi FOR EACH ROW DECLARE
   cc_ clienţi.codcl%TYPE ;
BEGIN
```

y

Ai o \oc

SELECT seq_clienti_codcl.NEXTVAL

```
   INTO cc
   FROM DUAL ;
   :NEW.codcl := cc_ ;
END ;
```

Declanşatoare pentru restricţii referenţiale

în Oracle, cea mai mare parte a restricţiilor referenţiale sunt rezolvate prin opţiunile comenzii `CREATE TABLE`. Astfel, ştergerea unei înregistrări-părinte, valoarea eronată a unei chei străine sunt sancţionate prompt de SGBD. Principala problemă referenţială rămasă nerezolvată în Oracle este actualizarea în cascadă a unui atribut-părinte în toate înregistrările-copil. Această opţiune (`UPDATE CASCADE`) poate fi implementată, cel puţin deocamdată, numai cu ajutorul declanşatoarelor.

Spre exemplu, creăm un declanşator pentru propagarea modificărilor atributului `Jud` din JUDEŢE în tabela LOCALITĂŢI - listingul 8.17.

Listing 8.17. Declanşator pentru modificarea atributului jud in tabela JUDEŢE

```
CREATE OR REPLACE TRIGGER trg_judete_upd_jud_after_row
AFTER UPDATE OF jud ON judeţe REFERENCING NEW AS NEW OLD
AS OLD FOR EACH ROW BEGIN
   UPDATE localitati SET jud = :NEW.jud WHERE jud = :OLD.jud ;
END ;
```

Atribute actualizate prin declanşatoare

Pentru comparaţie cu VFP, în listingul 8.18 este prezentat declanşatorul pentru modificarea unei linii a tabelei LINIIFACT, declanşator ce recalculează valoarea totală a facturii.

Listing 8.18. Crearea declanşatorului TRG_LINIIFACT_UPD_AFTER_ROW

```
CREATE OR REPLACE TRIGGER trg_liniifact_upd_after_row
AFTER UPDATE OF NrFact, CodPr, Cantitate, PretUnit ON
liniifact FOR EACH ROW DECLARE     ■
   v_proctva_OLD produse.proctva%TYPE ;
   v_proctva_NEW produse.proctva%TYPE ;
BEGIN
   variabile_globale.v_trg_liniifact_upd_after_row := TRUE ;
   SELECT ProcTVA INTO v_proctva_OLD
      FROM produse WHERE codpr = :OLD.codpr ;
   IF :NEW.codpr O :OLD.codpr THEN
      —  daca s-a schimbat codpr e posibil si procentul TVA
      —  sa fie altul
  *  SELECT ProcTVA INTO v_proctva_NEW FROM produse
         WHERE codpr = :NEW.codpr ;
  ELSE
      —  procent TVA neschimbat
      v_proctva_NEW := v_proctva_OLD
      ;
  END IF ;
  IF :NEW.nrfact <> :OLD.nrfact THEN
      —  s-a modificat nrfact, deci trebuie decrementată
```

```
   — Valtotala pt. factura veche    si incrementată
   — pentru factura noua
   UPDATE facturi SET valtotala = valtotala -
      :OLD.cantitate * :OLD.PretUnit *
      (1 + v proctva OLD) WHERE nrfact = :OLD.nrfact ;

   UPDATE facturi SET valtotala = valtotala +
      :NEW.cantitate * :NEW.pretunit *
      (1 + v proctva NEW) WHERE nrfact — :NEW.nrfact ;
ELSE
   UPDATE facturi
   SET valtotala = valtotala - :OLD.cantitate *
      :OLD.PretUnit *(1 + v_proctva_OLD) +
      :NEW.cantitate * :NEW.PretUnit *
      (1 + v_proctva_NEW)
   WHERE nrfact = :NEW.nrfact ;

END IF ;
variabile_globale.v_trg_liniifact_upd_after_row := FALSE ;
END ;
```

Ca o noutate apare şi variabila publică v_trg_liniifact_upd_after_row, care este `TRUE` numai pe parcursul declanşatorului. O variabilă are un regim public în PL/SQL prin intermediul unui pachet (dacă apare în specificaţiile pachetului). Listingul de creare a pachetului variabile_globale este prezentat peste câteva pagini.

Declanşatoare pentru controlul actualizărilor

Păstrând similitudinea cu declanşatoarele VFP, îl construim pe cel de modificare a tabelei FACTURI, în care trebuie să se interzică actualizarea atributului `ValTotala` **altfel decât prin declanşatorul** `trg_liniif act_upd_after_row`.

Listing 8.19. Prima variantă a declanşatorului TRG_FACTURI_UPD_AFTER_ROW

```
CREATE OR REPLACE TRIGGER
trg_facturi_upd_after_row AFTER UPDATE ON facturi FOR
EACH ROW DECLARE
v_vinzari facturi. valtotala%TYPE ; v
incasari facturi.valtotala%TYPE ;
```

```
BEGIN
   variabile_globale.v_trg_facturi_upd_after row := TRUE ;
   IF :NEW.vaitotala o :OLD.valtotala AND
         NOT variabile_globale.v_trg_liniifact_upd_after_row
      THEN
      variabile_globale.v_trg_facturi_upd_after_row := FALSE ;
      RAISE_AP PLICATION_ERROR(-20501,
         'Este interzisa modificarea directa a ValTotala !');
   END IF ;
   IF :NEW.codcl O :OLD.codcl THEN
      SELECT SUM (cantitate * pretunit * (1 + proctva))
      INTO v_vinzari
      FROM facturi f, liniifact lf, produse p WHERE
      f.nrfact=lf.nrfact AND lf.codpr=p.codpr AND
      codcl=:NEW.codcl ;
      SELECT SUM(transa)
      INTO v_incasari
      FROM facturi f, incasfact i
      WHERE f.nrfact=i.nrfact AND codcl = :NEW.codcl ;
      IF v_vinzari - v_incasari > 100000 THEN
         variabile_globale.v_trg_facturi_upd after row :=
            FALSE ;
         RAISE_APPLICATION_ERROR(-20502,
            'Clientul e prea mare datornic !');
      END IF ;
   END IF ;
   IF :NEW.nrfact o :0LD.nrfact THEN
      UPDATE liniifact SET nrfact = :NEW.nrfact WHERE nrfact =
         :OLD.nrfact ;
      UPDATE incasfact SET nrfact = :NEW.nrfact WHERE nrfact =
         :OLD.nrfact ;
   END IF ;
   variabile_globale.v_trg_facturi_upd_after_row := FALSE ;
END ;
```

Dacă la comanda

UPDATE facturi SET valtotala = 0 WHERE nrfact=llll ;

declanşatorul se comportă onorabil, afişând mesajul de eroare:

ERROR at line 1:
ORA-20501: Este interzisa modificarea directa a ValTotala ! ORA-06512: at "FOTACHEM.TRG_FACTURI_UPD_AFTER ROW", line 9 ORA-04 08 8: error during execution of trigger [1]FOTACHEM.TRG_FACTURI_UPD_AFTER_ROW'

în schimb, la modificarea lui CodCI pentru o factură:

UPDATE facturi SET codcl - 1002 WHERE nrfact = 1111 ;

facem cunoştinţă cu o problemă „clasică" din Oracle:

```
UPDATE facturi SET codcl = 1002 WHERE nrfact = 1111 *
ERROR at line 1:
ORA-04091: table FOTACHEM.FACTURI is mutating, trigger/function may not
see it ORA-O6512: at "FOTACHEM.TRG_FACTURI_UPD_AFTER_ROW", line
13 ORA-04088: error during execution of trigger
'FOTACHEM.TRG_FACTURI_UPD_AFTER_ROW'
```

Tabela FACTURI nu poate fi modificată deoarece, în termeni oracliști, este mutantă. O tabelă este mutantă (*mutating*) dacă se află în curs de modificare printr-o comandă INSERT, UPDATE sau DELETE sau trebuie modificată ca efect al unei restricţii referenţiale de tip DELETE

CASCADE. O tabelă este restricţionată (*constraining*) dacă declanşatorul o accesează direct, printr-o comandă SQL, sau indirect, în virtutea unei restricţii referenţiale.

Pentru toate declanşatoarele de la nivel de linie şi cel de la nivel de comandă (*statement*) lansat ca urmare a unei ştergeri în cascadă (prin opţiunea DELETE CASCADE de la CREATE/ALTER TABLE) există câteva restricţii menite să prevină situaţiile de inconsistenţă a datelor:

- într-un declanşator nu poate fi citită sau modificată o tabelă mutantă (în curs de modificare) datorită *trigger*-ului. Este ceea ce ni s-a întâmplat în FACTURI, când declanşatorul citeşte tabela pentru calculul valorilor facturate şi încasate ale noului client.'
- într-un *trigger* nu pot fi modificate atributele ce alcătuiesc cheile primară şi străine, precum şi cele pentru care a fost declarată restricţia de unicitate, atribute ale unei tabele restricţionate (pentru *trigger*-ul respectiv). Excepţie fac *trigger*-ele de inserare la nivel de linie prin care se adaugă o singură înregistrare.

Pentru a rezolva problema apărută, modificăm corpul declanşatorului de mai sus şi creăm un declanşator la nivel de comandă. Mai înainte, în listingul 8.20 este prezentat pachetul variabilelor publice.

Listing 8.20. Crearea pachetului cu variabile publice

```
CREATE OR REPLACE PACKAGE variabile_globale IS~
   v_trg_facturi_upd_befo_sta BOOLEAN := FALSE ;
   v_trg_facturi_upd_befo_row BOOLEAN := FALSE
   ;  v_trg_facturi_upd_after_row  BOOLEAN  :=
   FALSE ; v_trg_facturi_upd_after_sta BOOLEAN
   := FALSE ;
   v_trg_liniifact__upd_befo_sta BOOLEAN := FALSE
   ; v_trg_liniifact_upd_befo_row BOOLEAN : =
   FALSE ; v_trg_liniifact_upd_after_row BOOLEAN
   :=  FALSE  ;  v_trg_liniifact_upd_after_sta
   BOOLEAN := FALSE ;
   v_codcl_NEW clienţi.codcl%TYPE
   ; v_codcl_OLD
   clienţi.codcl%TYPE ;
END ;
```

în noua versiune a declanşatorului TRG_FACTURI_UPD_AFTER_ROW se memorează, în variabilele publice v_codcl_NEW şi v_codcl_OLD, vechea şi noua valoarea a atributului CodCI. Aceste două valori sunt folosite în noul declanşator, cel de la nivel de comandă, în care scăpăm de problema „mutanţei”:

Listing 8.21. Varianta 2 a declanşatorului TRG_FACTURI_UPD_AFTER_ROW

```
CREATE OR REPLACE TRIGGER trg_facturi_upd_after_row AFTER
UPDATE ON facturi FOR EACH ROW DECLARE
   v_vinzari facturi.valtotala%TYPE ;
   v_incasari facturi.valtotala%TYPE
   ;
BEGIN
   variabile_globale.v_trg_facturi_upd_after_row := TRUE ;
             variabile_globale.v_codcl_OLD := :OLD.codcl ;
             variabile_globale.v_codcl_NEW := :NEW.codcl ;
   IF :NEW.valtotala O :OLD.valtotala AND NOT
   variabile_globale.v_trg_liniifact_upd_after_row
   THEN
      variabile_globale.v_trg_facturi_upd_after_row := FALSE
```

```
      ; RAISE_APPLICATION_ERROR(-20501,
         'Este interzisa modificarea directa a ValTotala !');
   END IF ;
   IF :NEW.nrfact <> :OLD.nrfact THEN
      UPDATE liniifact SET nrfact = :NEW.nrfact
         WHERE nrfact = :OLD.nrfact ;
        UPDATE incasfact SET nrfact = :NEW.nrfact WHERE nrfact =
                              :OLD.nrfact ;
   END IF ;
   variabile_globale.v_trg_facturi_upd_after_row := FALSE ;
END ;
```

în final, listingul 8.22 conţine scriptul pentru crearea declanşatorului la nivel de comandă:

Listing 8.22. Crearea declanşatorului TRG_FACTURI_UPD_AFTER_STA

```
CREATE OR REPLACE TRIGGER trg_facturi_upd_after_sta AFTER
UPDATE ON facturi DECLARE
   v_vinzari facturi.valtotala%TYPE ;
   v_incasari facturi.valtotala%TYPE
   ;
BEGIN
   variabile_globale.v_trg_facturi_upd_after_sta := TRUE ;
   IF variabile_globale.v_codcl_OLD o
   variabile_globale.v_codcl_NEW THEN
      SELECT SUM (cantitate * pretunit * (1 + proctva))
      INTO v_vinzari
                 FROM facturi f, liniifact lf, produse p
     WHERE f.nrfact=lf.nrfact AND lf.codpr=p.codpr AND codcl =
        variabile_globale.v_codcl_NEW ;
     SELECT SUM(transa)
     INTO v_incasari FROM facturi f, incasfact i WHERE
     f.nrfact=i.nrfact AND codcl = variabile_globaie.v_codcl_NEW ;
     IF v vinzari - v_incasari > 10000000 THEN
        văriabile_globale.v_trg_facturi_upd_after_sta :=FALSE;
        RAISE_APPLICATION_ERROR(-20502,
           'Clientul e prea mare datornic !');
     END IF ;
     END IF ;
     variabile_globale.v_trg_facturi_upd_after_sta := FALSE ;
END ;
```

Declanşatoare de tip INSTEAD OF în Oracle 8i

După cum promiteam în paragraful dedicat tabelelor derivate, în Oracle 8i problemele legate de actualizarea acestora şi, implicit, propagarea modificărilor în tabelele persistente, se pot rezolva mai mult decât onorabil utilizând declanşatoare INSTEAD OF. Revenim la tabela virtuală vLocJudlasi creată prin joncţiunea tabelelor JUDEŢE şi LOCALITĂŢI. Inserarea unei linii în *view* înseamnă, în primul rând, adăugarea unei noi localităţi. în plus, trebuie verificat dacă valoarea atributului Jud există în JUDEŢE. Dacă nu, atunci este vorba despre un judeţ nou-nouţ şi se inserează o nouă linie în JUDEŢE. Iată, în listingul 8.23, blocul PL/SQL de creare a declanşatorului ce realizează aceste operaţiuni.

Listing 8.23. Bloc PL/SQL de creare a declanşatorului TRG_VLOCJUDIASI_INS_INSTEAD

```
CREATE OR REPLACE TRIGGER trg_vlocjudiasi_ins_instead
INSTEAD OF INSERT ON vlocjudiasi DECLARE
     jud_ JUDEŢE.Jud%TYPE ;                                          '
BEGIN
     BEGIN
     SELECT Jud INTO jud_
     FROM judeţe
     WHERE jud = :NEW.Jud ;
     EXCEPTION
     WHEN NO_DATA_FOUND THEN
          INSERT INTO judeţe VALUES (:NEW.Jud, :NEW.Judeţ,
             :NEW.Regiune) ;
     END ;
     INSERT INTO localitati VALUES (:NEW.CodPost, :NEW.Loc,
     :NEW.Jud) ;
```

Analog pot fi create declanşatoare pentru modificări şi ştergeri ale liniilor din tabela virtuală.

Toate informaţiile despre declanşatoarele tabelelor sau tabelelor virtuale din schema curentă, inclusiv corpul lor (blocul PL/SQL), pot fi obţinute din tabela `USER_TRIGGERS` **a catalogului-sistem. Spre exemplu, situaţia *trigger-e*lor tabelei LINIIFACT este vizualizată astfel:**

```
SELECT *
FROM USER_TRIGGERS
WHERE TABLE NAME = 'LINIIFACT'
```

CUPRINS

Cuvânt, oarecum, înainte 5

Conţin uitul lucrării.. 7

Capitolul 1 NOŢIUNI ALE MODELULUI RELAŢIONAL

1.1. Baze de date şi sisteme de gestiune a bazelor de date 9

Niveluri de abstractizare a datelor .....

Sisteme de gestiune a bazelor de date .....

Limbaje de definire a datelor .....

Limbaje de manipulare a datelor .....

Gestionarul bazei .....

Administratorul bazei de date .....

Utilizatorii bazelor de date .....

1.2. Un pic de istorie recentă şi relaţională 29

1.3. Relaţii/tabele, domenii şi atribute 24

1.4. Restricţii ale bazei de date .....

Restricţia de domeniu .....

Atonicitate .....

Restricţia de unicitate .....

Restricţia referenţială .....

Restricţii-utilizator .....

1.5. Schema şi conţinutul unei baze de date. Exemplu 34

Conentarii suplimentare privind restricţiile referenţiale 42

1.6. Alte noţiuni ale SGBD-urilor relaţionale 44

Tatele virtuale **(View-***uri)* .....

Proceduri stocate ....
..... * ....
..... 46

1.7. Regulile modelului relaţional..... 47

2. [Korth&Silberschatz88], pp. 3-5; [Everest86], pp. 31-34.
4. [Everest86], p. 11.
8. Preluare din [Korth&Silberschatz88], p. 19.
10. [Adiba&Delobel82], citaţi în [Saleh94], p. 24. Pentru detalii legate de obiectivele vizate de Codd atunci când a formulat modelul relaţional, vezi şi [Date99-2].
11. Pentru amănunte privind odiseea System R, vezi şi http://www.mcjones.org/System_R/SQL_Reunion_95.
13. Preluare din *PC World România*, iunie 1993, p. 33,
16. Pentru o discuţie *in extenso* a atomicităţii şi a primei forme normalizate, vezi [Fotache2000-1],
18. Beech, D. - „New Life for SQL", *Datamation*, 1 februarie 1989.
20. Vezi [Celko99], p. 49.
21. [Celko99], p. 52.
25. [Date86], pp. 90-91.
40. [Boyce ş.a. 75],
47. Preluare din [Hemandez&ViescaOO], pp. 62-63.

• O variantă SQL-92/DB2 este:

```
SELECT F. NrFact,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) /
      COUNT(DISTINCT VALUE( I.CodInc,1)) AS Facturat,
   SUM(VALUE(Transa,0))/ MAX(LF.Linie) AS încasat,
   SUM(Cantitate * PretUnit * (1+ProcTVA)) /
      COUNT(DISTINCT VALUE( I,Codlnc,1)) - SUM (VALUE
      (Transa, Cţ) ) / MAX (LF. Linie)
      AS Diferenţa,
   CASE WHEN SUM(VALUE(Transa,0))/ MAX(LF.Linie) = 0 THEN
      'Fara nici o incasare1
         WHEN SUM(Cantitate * PretUnit * (1+ProcTVA)) /
      COUNT(DISTINCT VALUE( I.Codlnc,1)) >
      SUM(VALUE(Transa,0))/ MAX(LF.Linie)
      THEN 'încasata parţial'
      ELSE 'ÎNCASATA TOTAL'
   END AS Situatiune
```

54. [Date86], p. 144.